

1 algorithm		
#include <algorithm> #include <numeric>		
Algo	Params	Funcion
sort, stable_sort	f, l	ordena el intervalo
nth_element	f, nth, l	<i>void</i> ordena el n-esimo, y particiona el resto
fill, fill_n	f, l / n, elem	<i>void</i> llena [f, l) o [f, f+n) con elem
lower_bound, upper_bound	f, l, elem	<i>it</i> al primer / ultimo donde se puede insertar elem para que quede ordenada
binary_search	f, l, elem	<i>bool</i> esta elem en [f, l)
copy	f, l, resul	hace resul+i=f+i $\forall i$
find, find_if, find_first_of	f, l, elem / pred / f2, l2	<i>it</i> encuentra i $\in [f, l)$ tq. i=elem, pred(i), $i \in [f2, l2)$
count, count_if	f, l, elem/pred	cuenta elem, pred(i)
search	f, l, f2, l2	busca $[f2, l2) \in [f, l)$
replace, replace_if	f, l, old / pred, new	cambia old / pred(i) por new
reverse	f, l	da vuelta
partition, stable_partition	f, l, pred	pred(i) ad, !pred(i) atras
min_element, max_element	f, l, [comp]	<i>it</i> min, max de [f, l]
lexicographical_compare	f1, l1, f2, l2	<i>bool</i> con $[f1, l1] \prec [f2, l2]$
next/prev_permutation	f, l	deja en [f, l) la perm sig, ant
set_intersection, set_difference, set_union, set_symmetric_difference,	f1, l1, f2, l2, res	[res, ...) la op. de conj
push_heap, pop_heap, make_heap	f, l, e / e /	mete/saca e en heap [f, l), hace un heap de [f, l)
is_heap	f, l	<i>bool</i> es [f, l) un heap
accumulate	f, l, i, [op]	$T = \sum$ /oper de [f, l)
inner_product	f1, l1, f2, i	$T = i + [f1, l1) \cdot [f2, \dots)$
partial_sum	f, l, r, [op]	$r+i = \sum$ /oper de $[f, f+i) \forall i \in [f, l)$
__builtin_ffs	unsigned int	Pos. del primer 1 desde la derecha
__builtin_clz	unsigned int	Cant. de ceros desde la izquierda.
__builtin_ctz	unsigned int	Cant. de ceros desde la derecha.
__builtin_popcount	unsigned int	Cant. de 1's en x.
__builtin_parity	unsigned int	1 si x es par, 0 si es impar.
__builtin_XXXXXXll	unsigned ll	= pero para long long's.

2 Estructuras

2.1 RMQ (static)

Dado un arreglo y una operacion asociativa *idempotente*, get(i, j) opera sobre el rango [i, j).  
Restriccion:  $LVL \geq \text{ceil}(\log n)$ ; Usar [ ] para llenar arreglo y luego build().

```
1 struct RMQ{
2     #define LVL 10
3     tipo vec[LVL][1<<(LVL+1)];
4     tipo &operator[] (int p){return vec[0][p];}
5     tipo get(int i, int j) {//intervalo [i,j)
6         int p = 31-__builtin_clz(j-i);
7         return min(vec[p][i], vec[p][j-(1<<p)]);
8     }
9     void build(int n) {//O(nlogn)
10        int mp = 31-__builtin_clz(n);
11        forn(p, mp) forn(x, n-(1<<p))
12            vec[p+1][x] = min(vec[p][x], vec[p][x+(1<<p)]);
13    };
```

2.2 RMQ (dynamic)

```
1 //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera sobre
2 el rango [i, j).
3 #define MAXN 100000
4 #define operacion(x, y) max(x, y)
5 const int neutro=0;
6 struct RMQ{
7     int sz;
8     tipo t[4*MAXN];
9     tipo &operator[] (int p){return t[sz+p];}
10    void init(int n){//O(nlgn)
11        sz = 1 << (32-__builtin_clz(n));
12        forn(i, 2*sz) t[i]=neutro;
13    }
14    void updall(){//O(n)
15        dforn(i, sz) t[i]=operacion(t[2*i], t[2*i+1]);}
16    tipo get(int i, int j){return get(i, j, 1, 0, sz);}
17    tipo get(int i, int j, int n, int a, int b){//O(lgn)
18        if(j<=a || i>=b) return neutro;
19        if(i<=a && b<=j) return t[n];
20        int c=(a+b)/2;
21        return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
22    }
23    void set(int p, tipo val){//O(lgn)
24        for(p+=sz; p>0 && t[p]!=val;){
25            t[p]=val;
```

```

25     p/=2;
26     val=operacion(t[p*2], t[p*2+1]);
27 }
28 }
29 }rmq;
30 //Usage:
31 cin >> n; rmq.init(n); for(n,i, n) cin >> rmq[i]; rmq.updall();

```

## 2.3 RMQ (lazy)

```

1 //Dado un arreglo y una operacion asociativa con neutro, get(i, j) opera sobre
  el rango [i, j].
2 typedef int Elem; //Elem de los elementos del arreglo
3 typedef int Alt; //Elem de la alteracion
4 #define operacion(x,y) x+y
5 const Elem neutro=0; const Alt neutro2=0;
6 #define MAXN 100000
7 struct RMQ{
8     int sz;
9     Elem t[4*MAXN];
10    Alt dirty[4*MAXN]; //las alteraciones pueden ser de distinto Elem
11    Elem &operator[](int p){return t[sz+p];}
12    void init(int n){//O(nlgn)
13        sz = 1 << (32-__builtin_clz(n));
14        for(n,i, 2*sz) t[i]=neutro;
15        for(n,i, 2*sz) dirty[i]=neutro2;
16    }
17    void push(int n, int a, int b){//propaga el dirty a sus hijos
18        if(dirty[n]!=0){
19            t[n]+=dirty[n]*(b-a); //altera el nodo
20            if(n<sz){
21                dirty[2*n]+=dirty[n];
22                dirty[2*n+1]+=dirty[n];
23            }
24            dirty[n]=0;
25        }
26    }
27    Elem get(int i, int j, int n, int a, int b){//O(lgn)
28        if(j<=a || i>=b) return neutro;
29        push(n, a, b); //corrige el valor antes de usarlo
30        if(i<=a && b<=j) return t[n];
31        int c=(a+b)/2;
32        return operacion(get(i, j, 2*n, a, c), get(i, j, 2*n+1, c, b));
33    }
34    Elem get(int i, int j){return get(i,j,1,0,sz);}
35    //altera los valores en [i, j] con una alteracion de val
36    void alterar(Alt val, int i, int j, int n, int a, int b){//O(lgn)

```

```

37    push(n, a, b);
38    if(j<=a || i>=b) return;
39    if(i<=a && b<=j){
40        dirty[n]+=val;
41        push(n, a, b);
42        return;
43    }
44    int c=(a+b)/2;
45    alterar(val, i, j, 2*n, a, c), alterar(val, i, j, 2*n+1, c, b);
46    t[n]=operacion(t[2*n], t[2*n+1]); //por esto es el push de arriba
47 }
48 void alterar(Alt val, int i, int j){alterar(val,i,j,1,0,sz);}
49 }rmq;

```

## 2.4 RMQ (persistente)

```

1 typedef int tipo;
2 tipo oper(const tipo &a, const tipo &b){
3     return a+b;
4 }
5 struct node{
6     tipo v; node *l,*r;
7     node(tipo v):v(v), l(NULL), r(NULL) {}
8     node(node *l, node *r) : l(l), r(r){
9         if(!l) v=r->v;
10        else if(!r) v=l->v;
11        else v=oper(l->v, r->v);
12    }
13 };
14 node *build (tipo *a, int tl, int tr) { //modificar para que tome tipo a
15     if (tl+1==tr) return new node(a[tl]);
16     int tm=(tl + tr)>>1;
17     return new node(build(a, tl, tm), build(a, tm, tr));
18 }
19 node *update(int pos, int new_val, node *t, int tl, int tr){
20     if (tl+1==tr) return new node(new_val);
21     int tm=(tl+tr)>>1;
22     if(pos < tm) return new node(update(pos, new_val, t->l, tl, tm), t->r);
23     else return new node(t->l, update(pos, new_val, t->r, tm, tr));
24 }
25 tipo get(int l, int r, node *t, int tl, int tr){
26     if(l==tl && tr==r) return t->v;
27     int tm=(tl + tr)>>1;
28     if(r<=tm) return get(l, r, t->l, tl, tm);
29     else if(l>=tm) return get(l, r, t->r, tm, tr);
30     return oper(get(l, tm, t->l, tl, tm), get(tm, r, t->r, tm, tr));
31 }

```

## 2.5 Fenwick Tree

```

1 //For 2D threat each column as a Fenwick tree, by adding a nested for in each
  operation
2 struct Fenwick{
3     static const int sz=1000001;
4     tipo t[sz];
5     void adjust(int p, tipo v){//valid with p in [1, sz), 0(lgn)
6         for(int i=p; i<sz; i+=(i&-i)) t[i]+=v; }
7     tipo sum(int p){//cumulative sum in [1, p], 0(lgn)
8         tipo s=0;
9         for(int i=p; i; i-=(i&-i)) s+=t[i];
10        return s;
11    }
12    tipo sum(int a, int b){return sum(b)-sum(a-1);}
13    //get largest value with cumulative sum less than or equal to x;
14    //for smallest, pass x-1 and add 1 to result
15    int getind(tipo x) { //0(lgn)
16        int idx = 0, mask = N;
17        while(mask && idx < N) {
18            int t = idx + mask;
19            if(x >= tree[t])
20                idx = t, x -= tree[t];
21            mask >>= 1;
22        }
23        return idx;
24    }
};

```

## 2.6 Union Find

```

1 struct UnionFind{
2     vector<int> f; //the array contains the parent of each node
3     void init(int n){f.clear(); f.insert(f.begin(), n, -1);}
4     int comp(int x){return (f[x]==-1?x:f[x]=comp(f[x]));} //0(1)
5     bool join(int i, int j) {
6         bool con=comp(i)==comp(j);
7         if(!con) f[comp(i)] = comp(j);
8         return con;
9     }
};

```

## 2.7 Disjoint Intervals

```

1 bool operator< (const ii &a, const ii &b) {return a.fst<b.fst;}
2 //Stores intervals as [first, second]
3 //in case of a collision it joins them in a single interval
4 struct disjoint_intervals {
5     set<ii> segs;
6     void insert(ii v) { //0(lgn)

```

```

7         if(v.snd-v.fst==0.) return; //0J0
8         set<ii>::iterator it, at;
9         at = it = segs.lower_bound(v);
10        if (at!=segs.begin() && (--at)->snd >= v.fst)
11            v.fst = at->fst, --it;
12        for(; it!=segs.end() && it->fst <= v.snd; segs.erase(it++))
13            v.snd=max(v.snd, it->snd);
14        segs.insert(v);
15    }
16 };

```

## 2.8 RMQ (2D)

```

1 struct RMQ2D{//n filas x m columnas
2     int sz;
3     RMQ t[4*MAXN];
4     RMQ &operator[] (int p){return t[sz/2+p];} //t[i][j]=i fila, j col
5     void init(int n, int m){ //0(n*m)
6         sz = 1 << (32-__builtin_clz(n));
7         forn(i, 2*sz) t[i].init(m); }
8     void set(int i, int j, tipo val){ //0(lgm.lgn)
9         for(i+=sz; i>0;){
10            t[i].set(j, val);
11            i/=2;
12            val=operacion(t[i*2][j], t[i*2+1][j]);
13        } }
14    tipo get(int i1, int j1, int i2, int j2){return get(i1,j1,i2,j2,1,0,sz);}
15    //0(lgm.lgn), rangos cerrado abierto
16    int get(int i1, int j1, int i2, int j2, int n, int a, int b){
17        if(i2<=a || i1>=b) return 0;
18        if(i1<=a && b<=i2) return t[n].get(j1, j2);
19        int c=(a+b)/2;
20        return operacion(get(i1, j1, i2, j2, 2*n, a, c),
21            get(i1, j1, i2, j2, 2*n+1, c, b));
22    }
23 } rmq;
24 //Example to initialize a grid of M rows and N columns:
25 RMQ2D rmq; rmq.init(n,m);
26 forn(i, n) forn(j, m){
27     int v; cin >> v; rmq.set(i, j, v);}

```

## 2.9 Big Int

```

1 #define BASEXP 6
2 #define BASE 1000000
3 #define LMAX 1000
4 struct bint{int l;ll n[LMAX];bint(ll x=0){l=1;forn(i,LMAX){if(x)l=i+1;n[i]=x%
    BASE;x/=BASE;}}bint(string x){l=(x.size()-1)/BASEXP+1;fill(n,n+LMAX,0);ll

```

```

r=1;forn(i,sz(x)){n[i/BASEXP]+=r*(x[x.size()-1-i]-'0');r*=10;if(r==BASE)r
=1;}}void out(){cout<<n[l-1];dforn(i,l-1)printf("%6.6llu",n[i]);}void
invar(){fill(n+l,n+LMAX,0);while(l>1&&!n[l-1])l--;};bint operator+(const
bint&a,const bint&b){bint c;c.l=max(a.l,b.l);ll q=0;forn(i,c.l)q+=a.n[i]+b
.n[i],c.n[i]=q/BASE,q/=BASE;if(q)c.n[c.l++] =q;c.invar();return c;}pair<
bint,bool>lresta(const bint&a,const bint&b){bint c;c.l=max(a.l,b.l);ll q
=0;forn(i,c.l)q+=a.n[i]-b.n[i],c.n[i]=(q+BASE)/BASE,q=(q+BASE)/BASE-1;c.
invar();return make_pair(c,!q);}bint&operator+=(bint&a,const bint&b){
return a=lresta(a,b).first;}bint operator-(const bint&a,const bint&b){
return lresta(a,b).first;}bool operator<(const bint&a,const bint&b){return
!lresta(a,b).second;}bool operator<=(const bint&a,const bint&b){return
lresta(b,a).second;}bool operator==(const bint&a,const bint&b){return a<=b
&&b<=a;}bint operator*(const bint&a,ll b){bint c;ll q=0;forn(i,a.l)q+=a.n[
i]*b,c.n[i]=q/BASE,q/=BASE;c.l=a.l;while(q)c.n[c.l++] =q/BASE,q/=BASE;c.
invar();return c;}bint operator*(const bint&a,const bint&b){bint c;c.l=a.l
+b.l;fill(c.n,c.n+b.l,0);forn(i,a.l){ll q=0;forn(j,b.l)q+=a.n[i]*b.n[j]+c.
n[i+j],c.n[i+j]=q/BASE,q/=BASE;c.n[i+b.l]=q;}c.invar();return c;}pair<bint
,ll>ldiv(const bint&a,ll b){bint c;ll rm=0;dforn(i,a.l){rm=rm*BASE+a.n[i];
c.n[i]=rm/b;rm%=b;}c.l=a.l;c.invar();return make_pair(c,rm);}bint operator
/(const bint&a,ll b){return ldiv(a,b).first;}ll operator%(const bint&a,ll
b){return ldiv(a,b).second;}pair<bint,bint>ldiv(const bint&a,const bint&b)
{bint c;bint rm=0;dforn(i,a.l){if(rm.l==1&&!rm.n[0])rm.n[0]=a.n[i];else{
dforn(j,rm.l)rm.n[j+1]=rm.n[j];rm.n[0]=a.n[i];rm.l++;}ll q=rm.n[b.l]*BASE+
rm.n[b.l-1];ll u=q/(b.n[b.l-1]+1);ll v=q/b.n[b.l-1]+1;while(u<v-1){ll m=(u
+v)/2;if(b*m<=rm)u=m;else v=m;}c.n[i]=u;rm-=b*u;}c.l=a.l;c.invar();return
make_pair(c,rm);}bint operator/(const bint&a,const bint&b){return ldiv(a,b
).first;}bint operator%(const bint&a,const bint&b){return ldiv(a,b).second
;}}

```

## 2.10 HashTables

```

1 //Compilar: g++ --std=c++11
2 struct Hash{
3     size_t operator()(const ii &a)const{
4         size_t s=hash<int>()(a.fst);
5         return hash<int>()(a.snd)+0x9e3779b9+(s<<6)+(s>>2);
6     }
7     size_t operator()(const vector<int> &v)const{
8         size_t s=0;
9         for(auto &e : v)
10             s ^= hash<int>()(e)+0x9e3779b9+(s<<6)+(s>>2);
11         return s;
12     }
13 };
14 unordered_set<ii, Hash> s;
15 unordered_map<ii, int, Hash> m; //map<key, value, hasher>

```

## 2.11 Modnum

```

1 struct mnum{
2     static const tipo mod=12582917;
3     tipo v;
4     mnum(tipo v=0): v(v%mod) {}
5     mnum operator+(mnum b){return v+b.v;}
6     mnum operator-(mnum b){return v>=b.v? v-b.v : mod-b.v+v;}
7     mnum operator*(mnum b){return v*b.v;}
8     mnum operator^(int n){
9         if(!n) return 1;
10        return n%2? (*this)^(n/2)*(this) : (*this)^(n/2);}
11 };

```

## 2.12 Treap para set

```

1 typedef int Key;
2 typedef struct node *pnode;
3 struct node{
4     Key key;
5     int prior, size;
6     pnode l,r;
7     node(Key key=0): key(key), prior(rand()), size(1), l(0), r(0) {}
8 };
9 static int size(pnode p) { return p ? p->size : 0; }
10 void push(pnode p) {
11     // modificar y propagar el dirty a los hijos aca(para lazy)
12 }
13 // Update function and size from children's Value
14 void pull(pnode p) { //recalcular valor del nodo aca (para rmq)
15     p->size = 1 + size(p->l) + size(p->r);
16 }
17 //junta dos arreglos
18 pnode merge(pnode l, pnode r) {
19     if (!l || !r) return l ? l : r;
20     push(l), push(r);
21     pnode t;
22     if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
23     else r->l=merge(l, r->l), t = r;
24     pull(t);
25     return t;
26 }
27 //parte el arreglo en dos, l<key<=r
28 void split(pnode t, Key key, pnode &l, pnode &r) {
29     if (!t) return void(l = r = 0);
30     push(t);
31     if (key <= t->key) split(t->l, key, l, t->l), r = t;
32     else split(t->r, key, t->r, r), l = t;

```

```

33     pull(t);
34 }
35
36 void erase(pnode &t, Key key) {
37     if (!t) return;
38     push(t);
39     if (key == t->key) t=merge(t->l, t->r);
40     else if (key < t->key) erase(t->l, key);
41     else erase(t->r, key);
42     if(t) pull(t);
43 }
44
45 ostream& operator<<(ostream &out, const pnode &t) {
46     if(!t) return out;
47     return out << t->l << t->key << ' ' << t->r;
48 }
49 pnode find(pnode t, Key key) {
50     if (!t) return 0;
51     if (key == t->key) return t;
52     if (key < t->key) return find(t->l, key);
53     return find(t->r, key);
54 }
55 struct treap {
56     pnode root;
57     treap(pnode root=0): root(root) {}
58     int size() { return ::size(root); }
59     void insert(Key key) {
60         pnode t1, t2; split(root, key, t1, t2);
61         t1::merge(t1, new node(key));
62         root::merge(t1, t2);
63     }
64     void erase(Key key1, Key key2) {
65         pnode t1, t2, t3;
66         split(root, key1, t1, t2);
67         split(t2, key2, t2, t3);
68         root=merge(t1, t3);
69     }
70     void erase(Key key) { ::erase(root, key); }
71     pnode find(Key key) { return ::find(root, key); }
72     Key &operator[](int pos){return find(pos)->key;}//ojito
73 };
74 treap merge(treap a, treap b) {return treap(merge(a.root, b.root));}

```

## 2.13 Treap para arreglo

```

1 typedef struct node *pnode;
2 struct node{

```

```

3     Value val, mini;
4     int dirty;
5     int prior, size;
6     pnode l, r, parent;
7     node(Value val): val(val), mini(val), dirty(0), prior(rand()), size(1), l
        (0), r(0), parent(0) {}
8 };
9 static int size(pnode p) { return p ? p->size : 0; }
10 void push(pnode p) { //propagar dirty a los hijos(aca para lazy)
11     p->val.fst+=p->dirty;
12     p->mini.fst+=p->dirty;
13     if(p->l) p->l->dirty+=p->dirty;
14     if(p->r) p->r->dirty+=p->dirty;
15     p->dirty=0;
16 }
17 static Value mini(pnode p) { return p ? push(p), p->mini : ii(1e9, -1); }
18 // Update function and size from children's Value
19 void pull(pnode p) { //recalcular valor del nodo aca (para rmq)
20     p->size = 1 + size(p->l) + size(p->r);
21     p->mini = min(min(p->val, mini(p->l)), mini(p->r)); //operacion del rmq!
22     p->parent=0;
23     if(p->l) p->l->parent=p;
24     if(p->r) p->r->parent=p;
25 }
26 //junta dos arreglos
27 pnode merge(pnode l, pnode r) {
28     if (!l || !r) return l ? l : r;
29     push(l), push(r);
30     pnode t;
31     if (l->prior < r->prior) l->r=merge(l->r, r), t = l;
32     else r->l=merge(l, r->l), t = r;
33     pull(t);
34     return t;
35 }
36 //parte el arreglo en dos, sz(l)==tam
37 void split(pnode t, int tam, pnode &l, pnode &r) {
38     if (!t) return void(l = r = 0);
39     push(t);
40     if (tam <= size(t->l)) split(t->l, tam, l, t->l), r = t;
41     else split(t->r, tam - 1 - size(t->l), t->r, r), l = t;
42     pull(t);
43 }
44 pnode at(pnode t, int pos) {
45     if(!t) exit(1);
46     push(t);
47     if(pos == size(t->l)) return t;

```

```

48 if(pos < size(t->l)) return at(t->l, pos);
49 return at(t->r, pos - 1 - size(t->l));
50 }
51 int getpos(pnode t){//inversa de at
52 if(!t->parent) return size(t->l);
53 if(t==t->parent->l) return getpos(t->parent)-size(t->r)-1;
54 return getpos(t->parent)+size(t->l)+1;
55 }
56 void split(pnode t, int i, int j, pnode &l, pnode &m, pnode &r) {
57 split(t, i, l, t), split(t, j-i, m, r);}
58 Value get(pnode &p, int i, int j){//like rmq
59 pnode l,m,r;
60 split(p, i, j, l, m, r);
61 Value ret=mini(m);
62 p=merge(l, merge(m, r));
63 return ret;
64 }
65 void print(const pnode &t) {//for debugging
66 if(!t) return;
67 push(t);
68 print(t->l);
69 cout << t->val.fst << '␣';
70 print(t->r);
71 }

```

## 2.14 Convex Hull Trick

```

1 struct Line{tipo m,h;};
2 tipo inter(Line a, Line b){
3     tipo x=b.h-a.h, y=a.m-b.m;
4     return x/y+(x%y?!((x>0)^(y>0)):0);//==ceil(x/y)
5 }
6 struct CHT {
7     vector<Line> c;
8     bool mx;
9     int pos;
10    CHT(bool mx=0):mx(mx),pos(0){};//mx=1 si las query devuelven el max
11    inline Line acc(int i){return c[c[0].m>c.back().m? i : sz(c)-1-i];}
12    inline bool irre(Line x, Line y, Line z){
13        return c[0].m>z.m? inter(y, z) <= inter(x, y)
14                : inter(y, z) >= inter(x, y);
15    }
16    void add(tipo m, tipo h) {//O(1), los m tienen que entrar ordenados
17        if(mx) m*=-1, h*=-1;
18        Line l=(Line){m, h};
19        if(sz(c) && m==c.back().m) { l.h=min(h, c.back().h), c.pop_back(); if(
20            pos) pos--; }

```

```

20 while(sz(c)>=2 && irre(c[sz(c)-2], c[sz(c)-1], l)) { c.pop_back(); if(
21     pos) pos--; }
22 c.pb(l);
23 }
24 inline bool fbin(tipo x, int m) {return inter(acc(m), acc(m+1))>x;}
25 tipo eval(tipo x){
26     int n = sz(c);
27     //query con x no ordenados O(lgn)
28     int a=-1, b=n-1;
29     while(b-a>1) { int m = (a+b)/2;
30         if(fbin(x, m)) b=m;
31         else a=m;
32     }
33     return (acc(b).m*x+acc(b).h)*(mx?-1:1);
34     //query O(1)
35     while(pos>0 && fbin(x, pos-1)) pos--;
36     while(pos<n-1 && !fbin(x, pos)) pos++;
37     return (acc(pos).m*x+acc(pos).h)*(mx?-1:1);
38 } ch;

```

## 2.15 Convex Hull Trick (Dynamic)

```

1 const ll is_query = -(1LL<<62);
2 struct Line {
3     ll m, b;
4     mutable multiset<Line>::iterator it;
5     const Line *succ(multiset<Line>::iterator it) const;
6     bool operator<(const Line& rhs) const {
7         if (rhs.b != is_query) return m < rhs.m;
8         const Line *s=succ(it);
9         if(!s) return 0;
10        ll x = rhs.m;
11        return b - s->b < (s->m - m) * x;
12    }
13 };
14 struct HullDynamic : public multiset<Line>{ // will maintain upper hull for
15     maximum
16     bool bad(iterator y) {
17         iterator z = next(y);
18         if (y == begin()) {
19             if (z == end()) return 0;
20             return y->m == z->m && y->b <= z->b;
21         }
22         iterator x = prev(y);
23         if (z == end()) return y->m == x->m && y->b <= x->b;
24         return (x->b - y->b)*(z->m - y->m) >= (y->b - z->b)*(y->m - x->m);

```



```

24 }
25 iterator next(iterator y){return ++y;}
26 iterator prev(iterator y){return --y;}
27 void insert_line(ll m, ll b) {
28     iterator y = insert((Line) { m, b });
29     y->it=y;
30     if (bad(y)) { erase(y); return; }
31     while (next(y) != end() && bad(next(y))) erase(next(y));
32     while (y != begin() && bad(prev(y))) erase(prev(y));
33 }
34 ll eval(ll x) {
35     Line l = *lower_bound((Line) { x, is_query });
36     return l.m * x + l.b;
37 }
38 }h;
39 const Line *Line::succ(multiset<Line>::iterator it) const{
40     return (++it==h.end())? NULL : &*it;};

```

## 2.16 Set con busq binaria

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4 typedef tree<int,null_type,less<int>,//key,mapped type, comparator
5     rb_tree_tag,tree_order_statistics_node_update> set_t;
6 //find_by_order(i) devuelve iterador al i-esimo elemento
7 //order_of_key(k): devuelve la pos del lower bound de k
8 //Ej: 12, 100, 505, 1000, 10000.
9 //order_of_key(10) == 0, order_of_key(100) == 1,
10 //order_of_key(707) == 3, order_of_key(9999999) == 5

```

## 3 Algos

### 3.1 Longest Increasing Subsequence

```

1 //Para non-increasing, cambiar comparaciones y revisar busq binaria
2 //Given an array, paint it in the least number of colors so that each color
   turns to a non-increasing subsequence.
3 //Solution:Min number of colors=Length of the longest increasing subsequence
4 int N, a[MAXN]; //secuencia y su longitud
5 ii d[MAXN+1]; //d[i]=ultimo valor de la subsecuencia de tamano i
6 int p[MAXN]; //padres
7 vector<int> R; //respuesta
8 void rec(int i){
9     if(i==1) return;
10    R.push_back(a[i]);
11    rec(p[i]);

```

```

12 }
13 int lis(){//O(nlogn)
14     d[0] = ii(-INF, -1); forn(i, N) d[i+1]=ii(INF, -1);
15     forn(i, N){
16         int j = upper_bound(d, d+N+1, ii(a[i], INF))-d;
17         if (d[j-1].first < a[i]&&a[i] < d[j].first){
18             p[i]=d[j-1].second;
19             d[j] = ii(a[i], i);
20         }
21     }
22     R.clear();
23     dforn(i, N+1) if(d[i].first!=INF){
24         rec(d[i].second); //reconstruir
25         reverse(R.begin(), R.end());
26         return i; //longitud
27     }
28     return 0;
29 }

```

## 3.2 Alpha-Beta pruning

```

1 ll alphabeta(State &s, bool player = true, int depth = 1e9, ll alpha = -INF, ll
   beta = INF) { //player = true -> Maximiza
2     if(s.isFinal()) return s.score;
3     //~ if (!depth) return s.heuristic();
4     vector<State> children;
5     s.expand(player, children);
6     int n = children.size();
7     forn(i, n) {
8         ll v = alphabeta(children[i], !player, depth-1, alpha, beta);
9         if(!player) alpha = max(alpha, v);
10        else beta = min(beta, v);
11        if(beta <= alpha) break;
12    }
13    return !player ? alpha : beta;}

```

## 3.3 Mo's algorithm

```

1 int n,sq;
2 struct Qu{//queries [l, r]
3     //intervalos cerrado abiertos !!! importante!!
4     int l, r, id;
5 }qs[MAXN];
6 int ans[MAXN], curans; //ans[i]=ans to ith query
7 bool bymos(const Qu &a, const Qu &b){
8     if(a.l/sq!=b.l/sq) return a.l<b.l;
9     return (a.l/sq)&1? a.r<b.r : a.r>b.r;
10 }

```

```

11 void mos(){
12     forn(i, t) qs[i].id=i;
13     sort(qs, qs+t, bymos);
14     int cl=0, cr=0;
15     sq=sqrt(n);
16     curans=0;
17     forn(i, t){ //intervalos cerrado abiertos !!! importante!!
18         Qu &q=qs[i];
19         while(cl>q.l) add(--cl);
20         while(cr<q.r) add(cr++);
21         while(cl<q.l) remove(cl++);
22         while(cr>q.r) remove(--cr);
23         ans[q.id]=curans;
24     }
25 }

```

## 4 Strings

### 4.1 Manacher

```

1 int d1[MAXN]; //d1[i]=long del maximo palindromo impar con centro en i
2 int d2[MAXN]; //d2[i]=analogo pero para longitud par
3 //0 1 2 3 4
4 //a a b c c <--d1[2]=3
5 //a a b b <--d2[2]=2 (estan uno antes)
6 void manacher(){
7     int l=0, r=-1, n=sz(s);
8     forn(i, n){
9         int k=(i>r? 1 : min(d1[l+r-i], r-i));
10        while(i+k<n && i-k>=0 && s[i+k]==s[i-k]) ++k;
11        d1[i] = k--;
12        if(i+k > r) l=i-k, r=i+k;
13    }
14    l=0, r=-1;
15    forn(i, n){
16        int k=(i>r? 0 : min(d2[l+r-i+1], r-i+1))+1;
17        while(i+k-1<n && i-k>=0 && s[i+k-1]==s[i-k]) k++;
18        d2[i] = --k;
19        if(i+k-1 > r) l=i-k, r=i+k-1;
20    }

```

### 4.2 KMP

```

1 string T; //cadena donde buscar(what)
2 string P; //cadena a buscar(what)
3 int b[MAXLEN]; //back table b[i] maximo borde de [0..i)
4 void kmppre(){ //by gabina with love

```

```

5     int i =0, j=-1; b[0]=-1;
6     while(i<sz(P)){
7         while(j>=0 && P[i] != P[j]) j=b[j];
8         i++, j++, b[i] = j;
9     }
10 }
11 void kmp(){
12     int i=0, j=0;
13     while(i<sz(T)){
14         while(j>=0 && T[i]!=P[j]) j=b[j];
15         i++, j++;
16         if(j==sz(P)) printf("P is found at index %d in T\n", i-j), j=b[j];
17     }
18 }

```

### 4.3 Trie

```

1 struct trie{
2     map<char, trie> m;
3     void add(const string &s, int p=0){
4         if(s[p]) m[s[p]].add(s, p+1);
5     }
6     void dfs(){
7         //Do stuff
8         forall(it, m)
9             it->second.dfs();
10    }
11 };

```

### 4.4 Suffix Array (largo, nlogn)

```

1 #define MAX_N 1000
2 #define rBOUND(x) (x<n? r[x] : 0)
3 //sa will hold the suffixes in order.
4 int sa[MAX_N], r[MAX_N], n;
5 string s; //input string, n=sz(s)
6
7 int f[MAX_N], tmpsa[MAX_N];
8 void countingSort(int k){
9     zero(f);
10    forn(i, n) f[rBOUND(i+k)]++;
11    int sum=0;
12    forn(i, max(255, n)){
13        int t=f[i]; f[i]=sum; sum+=t;
14    }
15    forn(i, n)
16        tmpsa[f[rBOUND(sa[i]+k)]++] = sa[i];
17    memcpy(sa, tmpsa, sizeof(sa));

```



```

18 void constructsa(){//O(n log n)
19     n=sz(s);
20     forn(i, n) sa[i]=i, r[i]=s[i];
21     for(int k=1; k<n; k<=<1){
22         countingSort(k), countingSort(0);
23         int rank, tmpr[MAX_N];
24         tmpr[sa[0]]=rank=0;
25         forr(i, 1, n)
26             tmpr[sa[i]]=(r[sa[i]]==r[sa[i-1]] && r[sa[i]+k]==r[sa[i-1]+k]) ? rank :
                ++rank;
27         memcpy(r, tmpr, sizeof(r));
28         if(r[sa[n-1]]==n-1) break;
29     }
30 }
31 void print(){//for debug
32     forn(i, n)
33         cout << i << ' ' <<
34         s.substr(sa[i], s.find( '$', sa[i])-sa[i]) << endl;

```

#### 4.5 String Matching With Suffix Array

```

1 //returns (lowerbound, upperbound) of the search
2 ii stringMatching(string P){ //O(sz(P)lgn)
3     int lo=0, hi=n-1, mid=lo;
4     while(lo<hi){
5         mid=(lo+hi)/2;
6         int res=s.compare(sa[mid], sz(P), P);
7         if(res>=0) hi=mid;
8         else lo=mid+1;
9     }
10    if(s.compare(sa[lo], sz(P), P)!=0) return ii(-1, -1);
11    ii ans; ans.fst=lo;
12    lo=0, hi=n-1, mid;
13    while(lo<hi){
14        mid=(lo+hi)/2;
15        int res=s.compare(sa[mid], sz(P), P);
16        if(res>0) hi=mid;
17        else lo=mid+1;
18    }
19    if(s.compare(sa[hi], sz(P), P)!=0) hi--;
20    ans.snd=hi;
21    return ans;
22 }

```

#### 4.6 LCP (Longest Common Prefix)

```

1 //Calculates the LCP between consecutives suffixes in the Suffix Array.
2 //LCP[i] is the length of the LCP between sa[i] and sa[i-1]

```

```

3 int LCP[MAX_N], phi[MAX_N], PLCP[MAX_N];
4 void computeLCP(){//O(n)
5     phi[sa[0]]=-1;
6     forr(i, 1, n) phi[sa[i]]=sa[i-1];
7     int L=0;
8     forn(i, n){
9         if(phi[i]==-1) {PLCP[i]=0; continue;}
10        while(s[i+L]==s[phi[i]+L]) L++;
11        PLCP[i]=L;
12        L=max(L-1, 0);
13    }
14    forn(i, n) LCP[i]=PLCP[sa[i]];
15 }

```

#### 4.7 Corasick

```

1
2 struct trie{
3     map<char, trie> next;
4     trie* tran[256]; //transiciones del automata
5     int idhoja, szhoja; //id de la hoja o 0 si no lo es
6     //link lleva al sufijo mas largo, nxthoja lleva al mas largo pero que es hoja
7     trie *padre, *link, *nxthoja;
8     char pch; //caracter que conecta con padre
9     trie(): tran(), idhoja(), padre(), link() {}
10    void insert(const string &s, int id=1, int p=0){ //id>0!!!
11        if(p<sz(s)){
12            trie &ch=next[s[p]];
13            tran[(int)s[p]]=&ch;
14            ch.padre=this, ch.pch=s[p];
15            ch.insert(s, id, p+1);
16        }
17        else idhoja=id, szhoja=sz(s);
18    }
19    trie* get_link() {
20        if(!link){
21            if(!padre) link=this; //es la raiz
22            else if(!padre->padre) link=padre; //hijo de la raiz
23            else link=padre->get_link()->get_tran(pch);
24        }
25        return link; }
26    trie* get_tran(int c) {
27        if(!tran[c]) tran[c] = !padre? this : this->get_link()->get_tran(c);
28        return tran[c]; }
29    trie *get_nxthoja(){
30        if(!nxthoja) nxthoja = get_link()->idhoja? link : link->nxthoja;
31        return nxthoja; }

```

```

32 void print(int p){
33     if(idhoja) cout << "found_" << idhoja << "_at_position_" << p-szhoja <<
        endl;
34     if(get_nxthoja()) get_nxthoja()->print(p); }
35 void matching(const string &s, int p=0){
36     print(p); if(p<sz(s)) get_tran(s[p])->matching(s, p+1); }
37 }tri;

```

## 4.8 Suffix Automaton

```

1 struct state {
2     int len, link;
3     map<char,int> next;
4     state() { }
5 };
6 const int MAXLEN = 10010;
7 state st[MAXLEN*2];
8 int sz, last;
9 void sa_init() {
10     forn(i,sz) st[i].next.clear();
11     sz = last = 0;
12     st[0].len = 0;
13     st[0].link = -1;
14     ++sz;
15 }
16 // Es un DAG de una sola fuente y una sola hoja
17 // cantidad de endpos = cantidad de apariciones = cantidad de caminos de la
    clase al nodo terminal
18 // cantidad de miembros de la clase = st[v].len-st[st[v].link].len (v>0) =
    caminos del inicio a la clase
19 // El arbol de los suffix links es el suffix tree de la cadena invertida. La
    string de la arista link(v)->v son los caracteres que difieren
20 void sa_extend (char c) {
21     int cur = sz++;
22     st[cur].len = st[last].len + 1;
23     // en cur agregamos la posicion que estamos extendiendo
24     //podria agregar tambien un identificador de las cadenas a las cuales
        pertenece (si hay varias)
25     int p;
26     for (p=last; p!=-1 && !st[p].next.count(c); p=st[p].link) // modificar esta
        linea para hacer separadores unicos entre varias cadenas (c=='$')
27         st[p].next[c] = cur;
28     if (p == -1)
29         st[cur].link = 0;
30     else {
31         int q = st[p].next[c];
32         if (st[p].len + 1 == st[q].len)

```

```

33         st[cur].link = q;
34     else {
35         int clone = sz++;
36         // no le ponemos la posicion actual a clone sino indirectamente por el
            link de cur
37         st[clone].len = st[p].len + 1;
38         st[clone].next = st[q].next;
39         st[clone].link = st[q].link;
40         for (; p!=-1 && st[p].next.count(c) && st[p].next[c]==q; p=st[p].link)
41             st[p].next[c] = clone;
42         st[q].link = st[cur].link = clone;
43     }
44 }
45 last = cur;
46 }

```

## 4.9 Z Function

```

1 char s[MAXN];
2 int z[MAXN]; // z[i] = i==0 ? 0 : max k tq s[0,k) match with s[i,i+k)
3 void z_function(char s[],int z[]) {
4     int n = strlen(s);
5     forn(i, n) z[i]=0;
6     for (int i = 1, l = 0, r = 0; i < n; ++i) {
7         if (i <= r) z[i] = min (r - i + 1, z[i - l]);
8         while (i + z[i] < n && s[z[i]] == s[i + z[i]]) ++z[i];
9         if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
10    }
11 }

```

# 5 Geometria

## 5.1 Punto

```

1 struct pto{
2     double x, y;
3     pto(double x=0, double y=0):x(x),y(y){}
4     pto operator+(pto a){return pto(x+a.x, y+a.y);}
5     pto operator-(pto a){return pto(x-a.x, y-a.y);}
6     pto operator+(double a){return pto(x+a, y+a);}
7     pto operator*(double a){return pto(x*a, y*a);}
8     pto operator/(double a){return pto(x/a, y/a);}
9     //dot product, producto interno:
10    double operator*(pto a){return x*a.x+y*a.y;}
11    //module of the cross product or vectorial product:
12    //if a is less than 180 clockwise from b, a^b>0
13    double operator^(pto a){return x*a.y-y*a.x;}

```

```

14 //returns true if this is at the left side of line qr
15 bool left(pto q, pto r){return ((q-*this)^(r-*this))>0;}
16 bool operator<(const pto &a) const{return x<a.x-EPS || (abs(x-a.x)<EPS && y<a
    .y-EPS);}
17 bool operator==(pto a){return abs(x-a.x)<EPS && abs(y-a.y)<EPS;}
18 double norm(){return sqrt(x*x+y*y);}
19 double norm_sq(){return x*x+y*y;}
20 };
21 double dist(pto a, pto b){return (b-a).norm();}
22 typedef pto vec;
23
24 double angle(pto a, pto o, pto b){
25     pto oa=a-o, ob=b-o;
26     return atan2(oa^ob, oa*ob);}
27
28 //rotate p by theta rads CCW w.r.t. origin (0,0)
29 pto rotate(pto p, double theta){
30     return pto(p.x*cos(theta)-p.y*sin(theta),
31         p.x*sin(theta)+p.y*cos(theta));
32 }

```

## 5.2 Orden radial de puntos

```

1 struct Cmp{//orden total de puntos alrededor de un punto r
2     pto r;
3     Cmp(pto r):r(r) {}
4     int cuad(const pto &a) const{
5         if(a.x > 0 && a.y >= 0)return 0;
6         if(a.x <= 0 && a.y > 0)return 1;
7         if(a.x < 0 && a.y <= 0)return 2;
8         if(a.x >= 0 && a.y < 0)return 3;
9         assert(a.x ==0 && a.y==0);
10        return -1;
11    }
12    bool cmp(const pto&p1, const pto&p2)const{
13        int c1 = cuad(p1), c2 = cuad(p2);
14        if(c1==c2) return p1.y*p2.x<p1.x*p2.y;
15        else return c1 < c2;
16    }
17    bool operator()(const pto&p1, const pto&p2) const{
18        return cmp(pto(p1.x-r.x,p1.y-r.y),pto(p2.x-r.x,p2.y-r.y));
19    }
20 };

```

## 5.3 Line

```

1 int sgn(ll x){return x<0? -1 : !!x;}
2 struct line{

```

```

3     line() {}
4     double a,b,c;//Ax+By=C
5     //pto MUST store float coordinates!
6     line(double a, double b, double c):a(a),b(b),c(c){}
7     line(pto p, pto q): a(q.y-p.y), b(p.x-q.x), c(a*p.x+b*p.y) {}
8     int side(pto p){return sgn(ll(a) * p.x + ll(b) * p.y - c);}
9 };
10 bool parallels(line l1, line l2){return abs(l1.a*l2.b-l2.a*l1.b)<EPS;}
11 pto inter(line l1, line l2){//intersection
12     double det=l1.a*l2.b-l2.a*l1.b;
13     if(abs(det)<EPS) return pto(INF, INF);//parallels
14     return pto(l2.b*l1.c-l1.b*l2.c, l1.a*l2.c-l2.a*l1.c)/det;
15 }

```

## 5.4 Segment

```

1 struct segm{
2     pto s,f;
3     segm(pto s, pto f):s(s), f(f) {}
4     pto closest(pto p) {//use for dist to point
5         double l2 = dist_sq(s, f);
6         if(l2==0.) return s;
7         double t =((p-s)*(f-s))/l2;
8         if (t<0.) return s;//not write if is a line
9         else if(t>1.)return f;//not write if is a line
10        return s+((f-s)*t);
11    }
12    bool inside(pto p){return abs(dist(s, p)+dist(p, f)-dist(s, f))<EPS;}
13 };
14
15 pto inter(segm s1, segm s2){
16     pto r=inter(line(s1.s, s1.f), line(s2.s, s2.f));
17     if(s1.inside(r) && s2.inside(r)) return r;
18     return pto(INF, INF);
19 }

```

## 5.5 Polygon Area

```

1 double area(vector<pto> &p){//O(sz(p))
2     double area=0;
3     forn(i, sz(p)) area+=p[i]^p[(i+1)%sz(p)];
4     //if points are in clockwise order then area is negative
5     return abs(area)/2;
6 }
7 //Area ellipse = M_PI*a*b where a and b are the semi axis lengths
8 //Area triangle = sqrt(s*(s-a)(s-b)(s-c)) where s=(a+b+c)/2

```

## 5.6 Circle

```

1  vec perp(vec v){return vec(-v.y, v.x);}
2  line bisector(pto x, pto y){
3      line l=line(x, y); pto m=(x+y)/2;
4      return line(-l.b, l.a, -l.b*m.x+l.a*m.y);
5  }
6  struct Circle{
7      pto o;
8      double r;
9      Circle(pto x, pto y, pto z){
10         o=inter(bisector(x, y), bisector(y, z));
11         r=dist(o, x);
12     }
13     pair<pto, pto> ptosTang(pto p){
14         pto m=(p+o)/2;
15         tipo d=dist(o, m);
16         tipo a=r*r/(2*d);
17         tipo h=sqrt(r*r-a*a);
18         pto m2=o+(m-o)*a/d;
19         vec per=perp(m-o)/d;
20         return make_pair(m2-per*h, m2+per*h);
21     }
22 };
23 //finds the center of the circle containing p1 and p2 with radius r
24 //as there may be two solutions swap p1, p2 to get the other
25 bool circle2PtsRad(pto p1, pto p2, double r, pto &c){
26     double d2=(p1-p2).norm_sq(), det=r*r/d2-0.25;
27     if(det<0) return false;
28     c=(p1+p2)/2+perp(p2-p1)*sqrt(det);
29     return true;
30 }
31 #define sqr(a) ((a)*(a))
32 #define feq(a,b) (fabs((a)-(b))<EPS)
33 pair<tipo, tipo> ecCuad(tipo a, tipo b, tipo c){//a*x*x+b*x+c=0
34     tipo dx = sqrt(b*b-4.0*a*c);
35     return make_pair((-b + dx)/(2.0*a), (-b - dx)/(2.0*a));
36 }
37 pair<pto, pto> interCL(Circle c, line l){
38     bool sw=false;
39     if((sw=feq(0,l.b))){
40         swap(l.a, l.b);
41         swap(c.o.x, c.o.y);
42     }
43     pair<tipo, tipo> rc = ecCuad(
44         sqr(l.a)+sqr(l.b),
45         2.0*l.a*l.b*c.o.y-2.0*(sqr(l.b)*c.o.x+l.c*l.a),
46         sqr(l.b)*(sqr(c.o.x)+sqr(c.o.y)-sqr(c.r))+sqr(l.c)-2.0*l.c*l.b*c.o.y

```

```

47     );
48     pair<pto, pto> p( pto(rc.first, (l.c - l.a * rc.first) / l.b),
49         pto(rc.second, (l.c - l.a * rc.second) / l.b) );
50     if(sw){
51         swap(p.first.x, p.first.y);
52         swap(p.second.x, p.second.y);
53     }
54     return p;
55 }
56 pair<pto, pto> interCC(Circle c1, Circle c2){
57     line l;
58     l.a = c1.o.x-c2.o.x;
59     l.b = c1.o.y-c2.o.y;
60     l.c = (sqr(c2.r)-sqr(c1.r)+sqr(c1.o.x)-sqr(c2.o.x)+sqr(c1.o.y)
61         -sqr(c2.o.y))/2.0;
62     return interCL(c1, l);
63 }

```

## 5.7 Point in Poly

```

1  //checks if v is inside of P, using ray casting
2  //works with convex and concave.
3  //excludes boundaries, handle it separately using segment.inside()
4  bool inPolygon(pto v, vector<pto>& P) {
5      bool c = false;
6      forn(i, sz(P)){
7          int j=(i+1)%sz(P);
8          if((P[j].y>v.y) != (P[i].y > v.y) &&
9              (v.x < (P[i].x - P[j].x) * (v.y-P[j].y) / (P[i].y - P[j].y) + P[j].x))
10             c = !c;
11     }
12     return c;
13 }

```

## 5.8 Point in Convex Poly log(n)

```

1  void normalize(vector<pto> &pt){//delete collinear points first!
2      //this makes it clockwise:
3      if(pt[2].left(pt[0], pt[1])) reverse(pt.begin(), pt.end());
4      int n=sz(pt), pi=0;
5      forn(i, n)
6          if(pt[i].x<pt[pi].x || (pt[i].x==pt[pi].x && pt[i].y<pt[pi].y))
7              pi=i;
8      vector<pto> shift(n);//puts pi as first point
9      forn(i, n) shift[i]=pt[(pi+i)%n];
10     pt.swap(shift);
11 }
12 bool inPolygon(pto p, const vector<pto> &pt){

```

```

13 //call normalize first!
14 if(p.left(pt[0], pt[1]) || p.left(pt[sz(pt)-1], pt[0])) return false;
15 int a=1, b=sz(pt)-1;
16 while(b-a>1){
17     int c=(a+b)/2;
18     if(!p.left(pt[0], pt[c])) a=c;
19     else b=c;
20 }
21 return !p.left(pt[a], pt[a+1]);
22 }

```

## 5.9 Convex Hull

```

1 //stores convex hull of P in S, CCW order
2 //left must return >=0 to delete collinear points!
3 void CH(vector<pto>& P, vector<pto> &S){
4     S.clear();
5     sort(P.begin(), P.end()); //first x, then y
6     forn(i, sz(P)){ //lower hull
7         while(sz(S)>= 2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
8         S.pb(P[i]);
9     }
10    S.pop_back();
11    int k=sz(S);
12    dforn(i, sz(P)){ //upper hull
13        while(sz(S) >= k+2 && S[sz(S)-1].left(S[sz(S)-2], P[i])) S.pop_back();
14        S.pb(P[i]);
15    }
16    S.pop_back();
17 }

```

## 5.10 Cut Polygon

```

1 //cuts polygon Q along the line ab
2 //stores the left side (swap a, b for the right one) in P
3 void cutPolygon(pto a, pto b, vector<pto> Q, vector<pto> &P){
4     P.clear();
5     forn(i, sz(Q)){
6         double left1=(b-a)^(Q[i]-a), left2=(b-a)^(Q[(i+1)%sz(Q)]-a);
7         if(left1>=0) P.pb(Q[i]);
8         if(left1*left2<0)
9             P.pb(inter(line(Q[i], Q[(i+1)%sz(Q)]), line(a, b)));
10    }
11 }

```

## 5.11 Bresenham

```

1 //plot a line approximation in a 2d map
2 void bresenham(pto a, pto b){

```

```

3 pto d=b-a; d.x=abs(d.x), d.y=abs(d.y);
4 pto s(a.x<b.x? 1: -1, a.y<b.y? 1: -1);
5 int err=d.x-d.y;
6 while(1){
7     m[a.x][a.y]=1; //plot
8     if(a==b) break;
9     int e2=err;
10    if(e2 >= 0) err-=2*d.y, a.x+=s.x;
11    if(e2 <= 0) err+= 2*d.x, a.y+= s.y;
12 }
13 }

```

## 5.12 Interseccion de Circulos en n3log(n)

```

1 struct event {
2     double x; int t;
3     event(double xx, int tt) : x(xx), t(tt) {}
4     bool operator <(const event &o) const { return x < o.x; }
5 };
6 typedef vector<Circle> VC;
7 typedef vector<event> VE;
8 int n;
9 double cuenta(VE &v, double A, double B) {
10     sort(v.begin(), v.end());
11     double res = 0.0, lx = ((v.empty())?0.0:v[0].x);
12     int contador = 0;
13     forn(i, sz(v)) {
14         //interseccion de todos (contador == n), union de todos (contador > 0)
15         //conjunto de puntos cubierto por exacta k Circulos (contador == k)
16         if (contador == n) res += v[i].x - lx;
17         contador += v[i].t, lx = v[i].x;
18     }
19     return res;
20 }
21 // Primitiva de sqrt(r*r - x*x) como funcion double de una variable x.
22 inline double primitiva(double x, double r) {
23     if (x >= r) return r*r*M_PI/4.0;
24     if (x <= -r) return -r*r*M_PI/4.0;
25     double raiz = sqrt(r*r-x*x);
26     return 0.5 * (x * raiz + r*r*atan(x/raiz));
27 }
28 double interCircle(VC &v) {
29     vector<double> p; p.reserve(v.size() * (v.size() + 2));
30     forn(i, sz(v)) p.push_back(v[i].c.x + v[i].r), p.push_back(v[i].c.x - v[i].r);
31     forn(i, sz(v)) forn(j, i) {
32         Circle &a = v[i], b = v[j];

```

```

33     double d = (a.c - b.c).norm();
34     if (fabs(a.r - b.r) < d && d < a.r + b.r) {
35         double alfa = acos((sqr(a.r) + sqr(d) - sqr(b.r)) / (2.0 * d * a.r)
36             );
37         pto vec = (b.c - a.c) * (a.r / d);
38         p.pb((a.c + rotate(vec, alfa)).x), p.pb((a.c + rotate(vec, -alfa)).
39             x);
40     }
41 }
42 sort(p.begin(), p.end());
43 double res = 0.0;
44 forn(i,sz(p)-1) {
45     const double A = p[i], B = p[i+1];
46     VE ve; ve.reserve(2 * v.size());
47     forn(j,sz(v)) {
48         const Circle &c = v[j];
49         double arco = primitiva(B-c.c.x,c.r) - primitiva(A-c.c.x,c.r);
50         double base = c.c.y * (B-A);
51         ve.push_back(event(base + arco,-1));
52         ve.push_back(event(base - arco, 1));
53     }
54     res += cuenta(ve,A,B);
55 }

```

## 6 Math

### 6.1 Identidades

$$\sum_{i=0}^n \binom{n}{i} = 2^n$$

$$\sum_{i=0}^n i \binom{n}{i} = n * 2^{n-1}$$

$$\sum_{i=m}^n i = \frac{n(n+1)}{2} - \frac{m(m-1)}{2} = \frac{(n+1-m)(n+m)}{2}$$

$$\sum_{i=0}^n i = \sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6}$$

$$\sum_{i=0}^n i(i-1) = \frac{8}{6} \left(\frac{n}{2}\right) \left(\frac{n}{2} + 1\right) (n+1) \text{ (doubles)} \rightarrow \text{Sino ver caso impar y par}$$

$$\sum_{i=0}^n i^3 = \left(\frac{n(n+1)}{2}\right)^2 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4} = \left[\sum_{i=1}^n i\right]^2$$

$$\sum_{i=0}^n i^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$$

$$\sum_{i=0}^n i^p = \frac{(n+1)^{p+1}}{p+1} + \sum_{k=1}^p \frac{B_k}{p-k+1} \binom{p}{k} (n+1)^{p-k+1}$$

$$r = e - v + k + 1$$

Teorema de Pick: (Area, puntos interiores y puntos en el borde)

$$A = I + \frac{B}{2} - 1$$

### 6.2 Ec. Caracteristica

$$a_0 T(n) + a_1 T(n-1) + \dots + a_k T(n-k) = 0$$

$$p(x) = a_0 x^k + a_1 x^{k-1} + \dots + a_k$$

Sean  $r_1, r_2, \dots, r_q$  las raíces distintas, de mult.  $m_1, m_2, \dots, m_q$

$$T(n) = \sum_{i=1}^q \sum_{j=0}^{m_i-1} c_{ij} n^j r_i^n$$

Las constantes  $c_{ij}$  se determinan por los casos base.

### 6.3 Combinatorio

```

1  forn(i, MAXN+1){//comb[i][k]=i tomados de a k
2      comb[i][0]=comb[i][i]=1;
3      forr(k, 1, i) comb[i][k]=(comb[i-1][k]+comb[i-1][k-1])%MOD;
4  }
5  ll lucas (ll n, ll k, int p){ //Calcula (n,k)%p teniendo comb[p][p]
6      //precalculado.
7      ll aux = 1;
8      while (n + k) aux = (aux * comb[n%p][k%p]) %p, n/=p, k/=p;
9      return aux;
10 }

```

### 6.4 Exp. de Numeros Mod.

```

1  ll expmod (ll b, ll e, ll m){//O(log b)
2      if(!e) return 1;
3      ll q= expmod(b,e/2,m); q=(q*q)%m;
4      return e%2? (b * q)%m : q;
5  }

```

### 6.5 Exp. de Matrices

```

1  #define SIZE 350
2  int NN;
3  double tmp[SIZE][SIZE];
4  void mul(double a[SIZE][SIZE], double b[SIZE][SIZE]){ zero(tmp);
5      forn(i, NN) forn(j, NN) forn(k, NN) res[i][j]+=a[i][k]*b[k][j];
6      forn(i, NN) forn(j, NN) a[i][j]=res[i][j];
7  }
8  void powmat(double a[SIZE][SIZE], int n, double res[SIZE][SIZE]){
9      forn(i, NN) forn(j, NN) res[i][j]=(i==j);
10     while(n){
11         if(n&1) mul(res, a), n--;
12         else mul(a, a), n/=2;
13     } }

```

### 6.6 Matrices y determinante $O(n^3)$

```

1  struct Mat {
2      vector<vector<double>> > vec;
3      Mat(int n): vec(n, vector<double>(n) ) {}
4      Mat(int n, int m): vec(n, vector<double>(m) ) {}
5      vector<double> &operator[](int f){return vec[f];}
6      const vector<double> &operator[](int f) const {return vec[f];}
7      int size() const {return sz(vec);}

```



```

8   Mat operator+(Mat &b) { ///this de n x m entonces b de n x m
9       Mat m(sz(b),sz(b[0]));
10      forn(i,sz(vec)) forn(j,sz(vec[0])) m[i][j] = vec[i][j] + b[i][j];
11      return m;    }
12  Mat operator*(const Mat &b) { ///this de n x m entonces b de m x t
13      int n = sz(vec), m = sz(vec[0]), t = sz(b[0]);
14      Mat mat(n,t);
15      forn(i,n) forn(j,t) forn(k,m) mat[i][j] += vec[i][k] * b[k][j];
16      return mat;    }
17  double determinant(){///sacado de e maxx ru
18      double det = 1;
19      int n = sz(vec);
20      Mat m(*this);
21      forn(i, n){///para cada columna
22          int k = i;
23          forr(j, i+1, n)///busco la fila con mayor val abs
24              if(abs(m[j][i])>abs(m[k][i])) k = j;
25          if(abs(m[k][i])<1e-9) return 0;
26          m[i].swap(m[k]);///la swapeo
27          if(i!=k) det = -det;
28          det *= m[i][i];
29          forr(j, i+1, n) m[i][j] /= m[i][i];
30          ///hago 0 todas las otras filas
31          forn(j, n) if (j!= i && abs(m[j][i])>1e-9)
32              forr(k, i+1, n) m[j][k]-=m[i][k]*m[j][i];
33      }
34      return det;
35  }
36 };

```

## 6.7 Teorema Chino del Resto

$$y = \sum_{j=1}^n (x_j * (\prod_{i=1, i \neq j}^n m_i)_{m_j}^{-1} * \prod_{i=1, i \neq j}^n m_i)$$

## 6.8 Criba

```

1  #define MAXP 100000 ///no necesariamente primo
2  int criba[MAXP+1];
3  void crearcriba(){
4      int w[] = {4,2,4,2,4,6,2,6};
5      for(int p=25;p<=MAXP;p+=10) criba[p]=5;
6      for(int p=9;p<=MAXP;p+=6) criba[p]=3;
7      for(int p=4;p<=MAXP;p+=2) criba[p]=2;
8      for(int p=7,cur=0;p*p<=MAXP;p+=w[cur++&7]) if (!criba[p])
9          for(int j=p*p;j<=MAXP;j+=(p<<1)) if(!criba[j]) criba[j]=p;
10 }

```

```

11 vector<int> primos;
12 void buscarprimos(){
13     crearcriba();
14     forr (i,2,MAXP+1) if (!criba[i]) primos.push_back(i);
15 }
16 ///~ Useful for bit trick: #define SET(i) ( criba[(i)>>5]|=1<<((i)&31) ), #
    define INDEX(i) ( (criba[i>>5]>>((i)&31))&1 ), unsigned int criba[MAXP
    /32+1];

```

## 6.9 Funciones de primos

Sea  $n = \prod p_i^{k_i}$ , fact(n) genera un map donde a cada  $p_i$  le asocia su  $k_i$

```

1  ///factoriza bien numeros hasta MAXP^2
2  map<ll,ll> fact(ll n){ ///0 (cant primos)
3      map<ll,ll> ret;
4      forall(p, primos){
5          while(!(n%p)){
6              ret[*p]++;///divisor found
7              n/=p;
8          }
9      }
10     if(n>1) ret[n]++;
11     return ret;
12 }
13 ///factoriza bien numeros hasta MAXP
14 map<ll,ll> fact2(ll n){ ///0 (lg n)
15     map<ll,ll> ret;
16     while (criba[n]){
17         ret[criba[n]]++;
18         n/=criba[n];
19     }
20     if(n>1) ret[n]++;
21     return ret;
22 }
23 ///Usar asi: divisores(fac, divs, fac.begin()); NO ESTA ORDENADO
24 void divisores(const map<ll,ll> &f, vector<ll> &divs, map<ll,ll>::iterator it,
25     ll n=1){
26     if(it==f.begin()) divs.clear();
27     if(it==f.end()) { divs.pb(n); return; }
28     ll p=it->fst, k=it->snd; ++it;
29     forn(_, k+1) divisores(f, divs, it, n), n*=p;
30 }
31 ll sumDiv (ll n){
32     ll rta = 1;
33     map<ll,ll> f=fact(n);
34     forall(it, f) {
35         ll pot = 1, aux = 0;

```

```

35   forn(i, it->snd+1) aux += pot, pot *= it->fst;
36   rta*=aux;
37   }
38   return rta;
39 }
40 ll eulerPhi (ll n){ // con criba: O(lg n)
41   ll rta = n;
42   map<ll,ll> f=fact(n);
43   forall(it, f) rta -= rta / it->first;
44   return rta;
45 }
46 ll eulerPhi2 (ll n){ // O (sqrt n)
47   ll r = n;
48   forr (i,2,n+1){
49     if ((ll)i*i > n) break;
50     if (n % i == 0){
51       while (n%i == 0) n/=i;
52       r -= r/i; }
53   }
54   if (n != 1) r-= r/n;
55   return r;
56 }

```

## 6.10 Phollard's Rho (rolando)

```

1 ll mulmod(ll a,ll b,ll c){ll x=0,y=a%c;while(b>0){if(b%2==1)x=(x+y)%c;y=(y*2)%c
; b/=2;}return x%c;}ll expmod(ll b,ll e,ll m){if(!e)return 1;ll q=expmod(b,
e/2,m);q=mulmod(q,q,m);return e%2?mulmod(b,q,m):q;}bool es_primo_prob(ll n
,int a){if(n==a)return true;ll s=0,d=n-1;while(d%2==0)s++,d/=2;ll x=expmod
(a,d,n);if((x==1)|| (x+1==n))return true;forn(i,s-1){x=mulmod(x,x,n);if(x
==1)return false;if(x+1==n)return true;}return false;}bool rabin(ll n){if(
n==1)return false;const int ar[]={2,3,5,7,11,13,17,19,23};forn(j,9)if(!
es_primo_prob(n,ar[j]))return false;return true;}ll rho(ll n){if((n&1)==0)
return 2;ll x=2,y=2,d=1;ll c=rand()%n+1;while(d==1){x=(mulmod(x,x,n)+c)%n;
y=(mulmod(y,y,n)+c)%n;y=(mulmod(y,y,n)+c)%n;if(x-y>0)d=gcd(x-y,n);else d=
gcd(y-x,n);}return d==n?rho(n):d;}map<ll,ll>prim;void factRho(ll n){if(n
==1)return;if(rabin(n)){prim[n]++;return;}ll factor=rho(n);factRho(factor)
;factRho(n/factor);}

```

## 6.11 GCD

```

1 tipo gcd(tipo a, tipo b){return a?gcd(b %a, a):b;}

```

## 6.12 Extended Euclid

```

1 void extendedEuclid (ll a, ll b){ //a * x + b * y = d
2   if (!b) { x = 1; y = 0; d = a; return;}
3   extendedEuclid (b, a%b);
4   ll x1 = y;

```

```

5   ll y1 = x - (a/b) * y;
6   x = x1; y = y1;
7 }

```

## 6.13 LCM

```

1 tipo lcm(tipo a, tipo b){return a / gcd(a,b) * b;}

```

## 6.14 Inversos

```

1 #define MAXMOD 15485867
2 ll inv[MAXMOD]; //inv[i]*i=1 mod MOD
3 void calc(int p){ //O(p)
4   inv[1]=1;
5   forr(i, 2, p) inv[i]= p-((p/i)*inv[p%i])%p;
6 }
7 int inverso(int x){ //O(log x)
8   return expmod(x, eulerphi(MOD)-2); //si mod no es primo(sacar a mano)
9   return expmod(x, MOD-2); //si mod es primo
10 }

```

## 6.15 Simpson

```

1 double integral(double a, double b, int n=10000) { //O(n), n=cantdiv
2   double area=0, h=(b-a)/n, fa=f(a), fb;
3   forn(i, n){
4     fb=f(a+h*(i+1));
5     area+=fa+ 4*f(a+h*(i+0.5)) +fb, fa=fb;
6   }
7   return area*h/6.;}

```

## 6.16 Fraction

```

1 tipo mcd(tipo a, tipo b){return a?mcd(b%a, a):b;}
2 struct frac{
3   tipo p,q;
4   frac(tipo p=0, tipo q=1):p(p),q(q) {norm();}
5   void norm(){
6     tipo a = mcd(p,q);
7     if(a) p/=a, q/=a;
8     else q=1;
9     if (q<0) q=-q, p=-p;}
10  frac operator+(const frac& o){
11    tipo a = mcd(q,o.q);
12    return frac(p*(o.q/a)+o.p*(q/a), q*(o.q/a));}
13  frac operator-(const frac& o){
14    tipo a = mcd(q,o.q);
15    return frac(p*(o.q/a)-o.p*(q/a), q*(o.q/a));}
16  frac operator*(frac o){

```

```

17     tipo a = mcd(q,o.p), b = mcd(o.q,p);
18     return frac((p/b)*(o.p/a), (q/a)*(o.q/b));}
19 frac operator/(frac o){
20     tipo a = mcd(q,o.q), b = mcd(o.p,p);
21     return frac((p/b)*(o.q/a), (q/a)*(o.p/b));}
22 bool operator<(const frac &o) const{return p*o.q < o.p*q;}
23 bool operator==(const frac o){return p==o.p&&q==o.q;}
24 };

```

## 6.17 Polinomio

```

1 struct poly {
2     vector<tipo> c;//guarda los coeficientes del polinomio
3     poly(const vector<tipo> &c): c(c) {}
4     poly() {}
5     bool isnull() {return c.empty();}
6     poly operator+(const poly &o) const {
7         int m = sz(c), n = sz(o.c);
8         vector<tipo> res(max(m,n));
9         for(i, m) res[i] += c[i];
10        for(i, n) res[i] += o.c[i];
11        return poly(res);    }
12    poly operator*(const tipo cons) const {
13        vector<tipo> res(sz(c));
14        for(i, sz(c)) res[i]=c[i]*cons;
15        return poly(res);    }
16    poly operator*(const poly &o) const {
17        int m = sz(c), n = sz(o.c);
18        vector<tipo> res(m+n-1);
19        for(i, m) for(j, n) res[i+j]+=c[i]*o.c[j];
20        return poly(res);    }
21    tipo eval(tipo v) {
22        tipo sum = 0;
23        dfor(i, sz(c)) sum=sum*v + c[i];
24        return sum; }
25    //poly contains only a vector<int> c (the coeficients)
26    //the following function generates the roots of the polynomial
27    //it can be easily modified to return float roots
28    set<tipo> roots(){
29        set<tipo> roots;
30        tipo a0 = abs(c[0]), an = abs(c[sz(c)-1]);
31        vector<tipo> ps,qs;
32        for(p,1,sqrt(a0)+1) if (a0%p==0) ps.pb(p),ps.pb(a0/p);
33        for(q,1,sqrt(an)+1) if (an%q==0) qs.pb(q),qs.pb(an/q);
34        forall(pt,ps)
35            forall(qt,qs) if ( (*pt) % (*qt)==0 ) {
36            tipo root = abs((*pt) / (*qt));

```

```

37        if (eval(root)==0) roots.insert(root);
38    }
39    return roots; }
40};
41pair<poly,tipo> ruffini(const poly p, tipo r) {
42    int n = sz(p.c) - 1 ;
43    vector<tipo> b(n);
44    b[n-1] = p.c[n];
45    dfor(k,n-1) b[k] = p.c[k+1] + r*b[k+1];
46    tipo resto = p.c[0] + r*b[0];
47    poly result(b);
48    return make_pair(result,resto);
49}
50poly interpolate(const vector<tipo>& x,const vector<tipo>& y) {
51    poly A; A.c.pb(1);
52    for(i,sz(x)) { poly aux; aux.c.pb(-x[i]), aux.c.pb(1), A = A * aux; }
53    poly S; S.c.pb(0);
54    for(i,sz(x)) { poly Li;
55        Li = ruffini(A,x[i]).fst;
56        Li = Li * (1.0 / Li.eval(x[i])); // here put a multiple of the coefficients
57        S = S + Li * y[i]; }
58    return S;
59}

```

## 6.18 Ec. Lineales

```

1 bool resolver_ev(Mat a, Vec y, Vec &x, Mat &ev){
2     int n = a.size(), m = n?a[0].size():0, rw = min(n, m);
3     vector<int> p; for(i,m) p.push_back(i);
4     for(i, rw) {
5         int uc=i, uf=i;
6         for(f, i, n) for(c, i, m) if(fabs(a[f][c])>fabs(a[uf][uc])) {uf=f;uc=c;}
7         if (freq(a[uf][uc], 0)) { rw = i; break; }
8         for(j, n) swap(a[j][i], a[j][uc]);
9         swap(a[i], a[uf]); swap(y[i], y[uf]); swap(p[i], p[uc]);
10        tipo inv = 1 / a[i][i]; //aca divide
11        for(j, i+1, n) {
12            tipo v = a[j][i] * inv;
13            for(k, i, m) a[j][k]-=v * a[i][k];
14            y[j] -= v*y[i];
15        }
16    } // rw = rango(a), aca la matriz esta triangulada
17    for(i, rw, n) if (!freq(y[i],0)) return false; // chequeo de compatibilidad
18    x = vector<tipo>(m, 0);
19    dfor(i, rw){
20        tipo s = y[i];

```

```

21     forr(j, i+1, rw) s -= a[i][j]*x[p[j]];
22     x[p[i]] = s / a[i][i]; //aca divide
23 }
24 ev = Mat(m-rw, Vec(m, 0)); // Esta parte va SOLO si se necesita el ev
25 forn(k, m-rw) {
26     ev[k][p[k+rw]] = 1;
27     dforn(i, rw){
28         tipo s = -a[i][k+rw];
29         forr(j, i+1, rw) s -= a[i][j]*ev[k][p[j]];
30         ev[k][p[i]] = s / a[i][i]; //aca divide
31     }
32 }
33 return true;
34 }

```

## 6.19 FFT

```

1 //~ typedef complex<double> base; //menos codigo, pero mas lento
2 //elegir si usar complejos de c (lento) o estos
3 struct base{
4     double r,i;
5     base(double r=0, double i=0):r(r), i(i){}
6     double real()const{return r;}
7     void operator/=(const int c){r/=c, i/=c;}
8 };
9 base operator*(const base &a, const base &b){
10     return base(a.r*b.r-a.i*b.i, a.r*b.i+a.i*b.r);}
11 base operator+(const base &a, const base &b){
12     return base(a.r+b.r, a.i+b.i);}
13 base operator-(const base &a, const base &b){
14     return base(a.r-b.r, a.i-b.i);}
15 vector<int> rev; vector<base> wlen_pw;
16 inline static void fft(base a[], int n, bool invert) {
17     forn(i, n) if(i<rev[i]) swap(a[i], a[rev[i]]);
18     for (int len=2; len<=n; len<=1) {
19         double ang = 2*M_PI/len * (invert?-1:+1);
20         int len2 = len>>1;
21         base wlen (cos(ang), sin(ang));
22         wlen_pw[0] = base (1, 0);
23         forr(i, 1, len2) wlen_pw[i] = wlen_pw[i-1] * wlen;
24         for (int i=0; i<n; i+=len) {
25             base t, *pu = a+i, *pv = a+i+len2, *pu_end = a+i+len2, *pw = &wlen_pw
                [0];
26             for (; pu!=pu_end; ++pu, ++pv, ++pw)
27                 t = *pv * *pw, *pv = *pu - t,*pu = *pu + t;
28         }
29     }

```

```

30     if (invert) forn(i, n) a[i]/= n;}
31 inline static void calc_rev(int n){//precalculo: llamar antes de fft!!
32     wlen_pw.resize(n), rev.resize(n);
33     int lg=31-__builtin_clz(n);
34     forn(i, n){
35         rev[i] = 0;
36         forn(k, lg) if(i&(1<<k)) rev[i]|=1<<(lg-1-k);
37     }}
38 inline static void multiply(const vector<int> &a, const vector<int> &b, vector<
    int> &res) {
39     vector<base> fa (a.begin(), a.end()), fb (b.begin(), b.end());
40     int n=1; while(n < max(sz(a), sz(b))) n <= 1; n <= 1;
41     calc_rev(n);
42     fa.resize (n), fb.resize (n);
43     fft (&fa[0], n, false), fft (&fb[0], n, false);
44     forn(i, n) fa[i] = fa[i] * fb[i];
45     fft (&fa[0], n, true);
46     res.resize(n);
47     forn(i, n) res[i] = int (fa[i].real() + 0.5); }
48 void toPoly(const string &s, vector<int> &P){//convierte un numero a polinomio
49     P.clear();
50     dforn(i, sz(s)) P.pb(s[i]-'0');}

```

## 6.20 Tablas y cotas (Primos, Divisores, Factoriales, etc)

Cantidad de primos menores que  $10^n$

$$\pi(10^1) = 4 ; \pi(10^2) = 25 ; \pi(10^3) = 168 ; \pi(10^4) = 1229 ; \pi(10^5) = 9592$$

$$\pi(10^6) = 78.498 ; \pi(10^7) = 664.579 ; \pi(10^8) = 5.761.455 ; \pi(10^9) = 50.847.534$$

$$\pi(10^{10}) = 455.052,511 ; \pi(10^{11}) = 4.118.054.813 ; \pi(10^{12}) = 37.607.912.018$$

### Divisores

Cantidad de divisores ( $\sigma_0$ ) para *algunos*  $n/\neg\exists n' < n, \sigma_0(n') \geq \sigma_0(n)$

$$\sigma_0(60) = 12 ; \sigma_0(120) = 16 ; \sigma_0(180) = 18 ; \sigma_0(240) = 20 ; \sigma_0(360) = 24$$

$$\sigma_0(720) = 30 ; \sigma_0(840) = 32 ; \sigma_0(1260) = 36 ; \sigma_0(1680) = 40 ; \sigma_0(10080) = 72$$

$$\sigma_0(15120) = 80 ; \sigma_0(50400) = 108 ; \sigma_0(83160) = 128 ; \sigma_0(110880) = 144$$

$$\sigma_0(498960) = 200 ; \sigma_0(554400) = 216 ; \sigma_0(1081080) = 256 ; \sigma_0(1441440) = 288 \quad \sigma_0(4324320) = 384 ; \sigma_0(8648640) = 448$$

## 7 Grafos

### 7.1 Dijkstra

```

1 #define add(a, b, w) G[a].pb(make_pair(w, b))
2 ll dijkstra(int s, int t){//O(|E| log |V|)
3     priority_queue<ii, vector<ii>, greater<ii> > Q;
4     vector<ll> dist(N, INF); vector<int> dad(N, -1);
5     Q.push(make_pair(0, s)); dist[s] = 0;

```

```

6   while(sz(Q)){
7       ii p = Q.top(); Q.pop();
8       if(p.snd == t) break;
9       forall(it, G[p.snd])
10          if(dist[p.snd]+it->first < dist[it->snd]){
11              dist[it->snd] = dist[p.snd] + it->fst;
12              dad[it->snd] = p.snd;
13              Q.push(make_pair(dist[it->snd], it->snd)); }
14   }
15   return dist[t];
16   if(dist[t]<INF)//path generator
17       for(int i=t; i!=-1; i=dad[i])
18           printf("%d%c", i, (i==s?'\\n':'\\u'));}

```

## 7.2 Bellman-Ford

```

1  vector<ii> G[MAX_N]; //ady. list with pairs (weight, dst)
2  int dist[MAX_N];
3  void bford(int src){ //O(VE)
4      dist[src]=0;
5      forn(i, N-1) forn(j, N) if(dist[j]!=INF) forall(it, G[j])
6          dist[it->snd]=min(dist[it->snd], dist[j]+it->fst);
7  }
8
9  bool hasNegCycle(){
10     forn(j, N) if(dist[j]!=INF) forall(it, G[j])
11         if(dist[it->snd]>dist[j]+it->fst) return true;
12     //inside if: all points reachable from it->snd will have -INF distance(do bfs
13     )
14     return false;
15 }

```

## 7.3 Floyd-Warshall

```

1  //G[i][j] contains weight of edge (i, j) or INF
2  //G[i][i]=0
3  int G[MAX_N][MAX_N];
4  void floyd(){ //O(N^3)
5      forn(k, N) forn(i, N) if(G[i][k]!=INF) forn(j, N) if(G[k][j]!=INF)
6          G[i][j]=min(G[i][j], G[i][k]+G[k][j]);
7  }
8  bool inNegCycle(int v){
9      return G[v][v]<0;}
10 //checks if there's a neg. cycle in path from a to b
11 bool hasNegCycle(int a, int b){
12     forn(i, N) if(G[a][i]!=INF && G[i][i]<0 && G[i][b]!=INF)
13         return true;
14     return false;

```

```

15 }

```

## 7.4 Kruskal

```

1  struct Ar{int a,b,w;};
2  bool operator<(const Ar& a, const Ar &b){return a.w<b.w;}
3  vector<Ar> E;
4  ll kruskal(){
5      ll cost=0;
6      sort(E.begin(), E.end()); //ordenar aristas de menor a mayor
7      uf.init(n);
8      forall(it, E){
9          if(uf.comp(it->a)!=uf.comp(it->b)){ //si no estan conectados
10              uf.unir(it->a, it->b); //conectar
11              cost+=it->w;
12          }
13      }
14      return cost;
15 }

```

## 7.5 Prim

```

1  bool taken[MAXN];
2  priority_queue<ii, vector<ii>, greater<ii> > pq; //min heap
3  void process(int v){
4      taken[v]=true;
5      forall(e, G[v])
6          if(!taken[e->second]) pq.push(*e);
7  }
8
9  ll prim(){
10     zero(taken);
11     process(0);
12     ll cost=0;
13     while(sz(pq)){
14         ii e=pq.top(); pq.pop();
15         if(!taken[e.second]) cost+=e.first, process(e.second);
16     }
17     return cost;
18 }

```

## 7.6 2-SAT + Tarjan SCC

```

1  //We have a vertex representing a var and other for his negation.
2  //Every edge stored in G represents an implication. To add an equation of the
   form a||b, use addor(a, b)
3  //MAX=max cant var, n=cant var
4  #define addor(a, b) (G[neg(a)].pb(b), G[neg(b)].pb(a))
5  vector<int> G[MAX*2];

```

```

6 //idx[i]=index assigned in the dfs
7 //lw[i]=lowest index(closer from the root) reachable from i
8 int lw[MAX*2], idx[MAX*2], qidx;
9 stack<int> q;
10 int qcmp, cmp[MAX*2];
11 //verdad[cmp[i]]=valor de la variable i
12 bool verdad[MAX*2+1];
13
14 int neg(int x) { return x>=n? x-n : x+n;}
15 void tjn(int v){
16     lw[v]=idx[v]==qidx;
17     q.push(v), cmp[v]--;
18     forall(it, G[v]){
19         if(!idx[*it] || cmp[*it]==-2){
20             if(!idx[*it]) tjn(*it);
21             lw[v]=min(lw[v], lw[*it]);
22         }
23     }
24     if(lw[v]==idx[v]){
25         int x;
26         do{x=q.top(); q.pop(); cmp[x]=qcmp;}while(x!=v);
27         verdad[qcmp]=(cmp[neg(v)]<0);
28         qcmp++;
29     }
30 }
31 //remember to CLEAR G!!!
32 bool satisf(){//O(n)
33     memset(idx, 0, sizeof(idx)), qidx=0;
34     memset(cmp, -1, sizeof(cmp)), qcmp=0;
35     forn(i, n){
36         if(!idx[i]) tjn(i);
37         if(!idx[neg(i)]) tjn(neg(i));
38     }
39     forn(i, n) if(cmp[i]==cmp[neg(i)]) return false;
40     return true;
41 }

```

## 7.7 Articulation Points

```

1 int N;
2 vector<int> G[1000000];
3 //V[i]=node number(if visited), L[i]= lowest V[i] reachable from i
4 int qV, V[1000000], L[1000000], P[1000000];
5 void dfs(int v, int f){
6     L[v]=V[v]==qV;
7     forall(it, G[v])
8         if(!V[*it]){

```

```

9         dfs(*it, v);
10        L[v] = min(L[v], L[*it]);
11        P[v]+= L[*it]>=V[v];
12    }
13    else if(*it!=f)
14        L[v]=min(L[v], V[*it]);
15 }
16 int cantart(){ //O(n)
17     qV=0;
18     zero(V), zero(P);
19     dfs(1, 0); P[1]--;
20     int q=0;
21     forn(i, N) if(P[i]) q++;
22 return q;
23 }

```

## 7.8 Comp. Biconexas y Puentes

```

1 struct edge {
2     int u,v, comp;
3     bool bridge;
4 };
5 vector<edge> e;
6 void addEdge(int u, int v) {
7     G[u].pb(sz(e)), G[v].pb(sz(e));
8     e.pb((edge){u,v,-1,false});
9 }
10 //d[i]=id de la dfs
11 //b[i]=lowest id reachable from i
12 int d[MAXN], b[MAXN], t;
13 int nbc;//cant componentes
14 int comp[MAXN];//comp[i]=cant comp biconexas a la cual pertenece i
15 void initDfs(int n) {
16     zero(G), zero(comp);
17     e.clear();
18     forn(i,n) d[i]=-1;
19     nbc = t = 0;
20 }
21 stack<int> st;
22 void dfs(int u, int pe) { //O(n + m)
23     b[u] = d[u] = t++;
24     comp[u] = (pe != -1);
25     forall(ne, G[u]) if (*ne != pe){
26         int v = e[*ne].u ^ e[*ne].v ^ u;
27         if (d[v] == -1) {
28             st.push(*ne);
29             dfs(v,*ne);

```



```

30     if (b[v] > d[u]){
31         e[*ne].bridge = true; // bridge
32     }
33     if (b[v] >= d[u]){ // art
34         int last;
35         do {
36             last = st.top(); st.pop();
37             e[last].comp = nbc;
38         } while (last != *ne);
39         nbc++;
40         comp[u]++;
41     }
42     b[u] = min(b[u], b[v]);
43 }
44 else if (d[v] < d[u]) { // back edge
45     st.push(*ne);
46     b[u] = min(b[u], d[v]);
47 }
48 }
49 }

```

## 7.9 LCA + Climb

```

1  const int MAXN=100001;
2  const int LOGN=20;
3  //f[v][k] holds the 2^k father of v
4  //L[v] holds the level of v
5  int N, f[MAXN][LOGN], L[MAXN];
6  //call before build:
7  void dfs(int v, int fa=-1, int lvl=0){//generate required data
8      f[v][0]=fa, L[v]=lvl;
9      forall(it, G[v])if(*it!=fa) dfs(*it, v, lvl+1); }
10 void build(){//f[i][0] must be filled previously, 0(nlgn)
11     forn(k, LOGN-1) forn(i, N) f[i][k+1]=f[f[i][k]][k];}
12 #define lg(x) (31-__builtin_clz(x))//=floor(log2(x))
13 int climb(int a, int d){//O(lgn)
14     if(!d) return a;
15     dforn(i, lg(L[a])+1) if(1<<i<=d) a=f[a][i], d-=1<<i;
16     return a;}
17 int lca(int a, int b){//O(lgn)
18     if(L[a]<L[b]) swap(a, b);
19     a=climb(a, L[a]-L[b]);
20     if(a==b) return a;
21     dforn(i, lg(L[a])+1) if(f[a][i]!=f[b][i]) a=f[a][i], b=f[b][i];
22     return f[a][0]; }
23 int dist(int a, int b) {//returns distance between nodes
24     return L[a]+L[b]-2*L[lca(a, b)];}

```

## 7.10 Heavy Light Decomposition

```

1  int treesz[MAXN];//cantidad de nodos en el subarbol del nodo v
2  int dad[MAXN];//dad[v]=padre del nodo v
3  void dfs1(int v, int p=-1){//pre-dfs
4      dad[v]=p;
5      treesz[v]=1;
6      forall(it, G[v]) if(*it!=p){
7          dfs1(*it, v);
8          treesz[v]+=treesz[*it];
9      }
10 }
11 //PONER Q EN 0 !!!!
12 int pos[MAXN], q;//pos[v]=posicion del nodo v en el recorrido de la dfs
13 //Las cadenas aparecen continuas en el recorrido!
14 int cantcad;
15 int homecad[MAXN];//dada una cadena devuelve su nodo inicial
16 int cad[MAXN];//cad[v]=cadena a la que pertenece el nodo
17 void heavylight(int v, int cur=-1){
18     if(cur==-1) homecad[cur=cantcad++]=v;
19     pos[v]=q++;
20     cad[v]=cur;
21     int mx=-1;
22     forn(i, sz(G[v])) if(G[v][i]!=dad[v])
23         if(mx==-1 || treesz[G[v][mx]]<treesz[G[v][i]]) mx=i;
24     if(mx!=-1) heavylight(G[v][mx], cur);
25     forn(i, sz(G[v])) if(i!=mx && G[v][i]!=dad[v])
26         heavylight(G[v][i], -1);
27 }
28 //ejemplo de obtener el maximo numero en el camino entre dos nodos
29 //RTA: max(query(low, u), query(low, v)), con low=lca(u, v)
30 //esta funcion va trepando por las cadenas
31 int query(int an, int v){//O(logn)
32     //si estan en la misma cadena:
33     if(cad[an]==cad[v]) return rmq.get(pos[an], pos[v]+1);
34     return max(query(an, dad[homecad[cad[v]]]),
35               rmq.get(pos[homecad[cad[v]]], pos[v]+1));
36 }

```

## 7.11 Centroid Decomposition

```

1  int n;
2  vector<int> G[MAXN];
3  bool taken[MAXN];//poner todos en FALSE al principio!!
4  int padre[MAXN];//padre de cada nodo en el centroid tree
5
6  int szt[MAXN];
7  void calcsz(int v, int p) {

```

```

8   szt[v] = 1;
9   forall(it,G[v]) if (*it!=p && !taken[*it])
10      calcsz(*it,v), szt[v]+=szt[*it];
11 }
12 void centroid(int v=0, int f=-1, int lvl=0, int tam=-1) { //O(nlogn)
13     if(tam==-1) calcsz(v, -1), tam=szt[v];
14     forall(it, G[v]) if(!taken[*it] && szt[*it]>=tam/2)
15         {szt[v]=0; centroid(*it, f, lvl, tam); return;}
16     taken[v]=true;
17     padre[v]=f;
18     forall(it, G[v]) if(!taken[*it])
19         centroid(*it, v, lvl+1, -1);
20 }

```

## 7.12 Euler Cycle

```

1  int n,m,ars[MAXE], eq;
2  vector<int> G[MAXN]; //fill G,n,m,ars,eq
3  list<int> path;
4  int used[MAXN];
5  bool usede[MAXE];
6  queue<list<int>::iterator> q;
7  int get(int v){
8      while(used[v]<sz(G[v]) && usede[ G[v][used[v]] ]) used[v]++;
9      return used[v];
10 }
11 void explore(int v, int r, list<int>::iterator it){
12     int ar=G[v][get(v)]; int u=v^ars[ar];
13     usede[ar]=true;
14     list<int>::iterator it2=path.insert(it, u);
15     if(u!=r) explore(u, r, it2);
16     if(get(v)<sz(G[v])) q.push(it);
17 }
18 void euler(){
19     zero(used), zero(usede);
20     path.clear();
21     q=queue<list<int>::iterator>();
22     path.push_back(0); q.push(path.begin());
23     while(sz(q)){
24         list<int>::iterator it=q.front(); q.pop();
25         if(used[*it]<sz(G[*it])) explore(*it, *it, it);
26     }
27     reverse(path.begin(), path.end());
28 }
29 void addEdge(int u, int v){
30     G[u].pb(eq), G[v].pb(eq);
31     ars[eq++]=u^v;

```

```

32 }

7.13 Diametro árbol

1  vector<int> G[MAXN]; int n,m,p[MAXN],d[MAXN],d2[MAXN];
2  int bfs(int r, int *d) {
3      queue<int> q;
4      d[r]=0; q.push(r);
5      int v;
6      while(sz(q)) { v=q.front(); q.pop();
7          forall(it,G[v]) if (d[*it]==-1)
8              d[*it]=d[v]+1, p[*it]=v, q.push(*it);
9      }
10     return v; //ultimo nodo visitado
11 }
12 vector<int> diams; vector<ii> centros;
13 void diametros(){
14     memset(d,-1,sizeof(d));
15     memset(d2,-1,sizeof(d2));
16     diams.clear(), centros.clear();
17     forn(i, n) if(d[i]==-1){
18         int v,c;
19         c=bfs(bfs(i, d2), d);
20         forn(_,d[v]/2) c=p[c];
21         diams.pb(d[v]);
22         if(d[v]&1) centros.pb(ii(c, p[c]));
23         else centros.pb(ii(c, c));
24     }
25 }

```

## 7.14 Chu-liu

```

1  void visit(graph&h,int v,int s,int r,vector<int>&no,vector<vector<int>>&comp,
    vector<int>&prev,vector<vector<int>>&next,vector<weight>&mcost,vector<int>
    >&mark,weight&cost,bool&found){if(mark[v]){vector<int>temp=no;found=true;
    do{cost+=mcost[v];v=prev[v];if(v!=s){while(comp[v].size()>0){no[comp[v].
    back()]=s;comp[s].push_back(comp[v].back());comp[v].pop_back();}}while(v
    !=s);forall(j,comp[s])if(*j!=r)forall(e,h[*j])if(no[e->src]!=s)e->w-=mcost
    [ temp[*j] ];}mark[v]=true;forall(i,next[v])if(no[*i]!=no[v]&&prev[no[*i]
    ]==v)if(!mark[no[*i]]||*i==s)visit(h,*i,s,r,no,comp,prev,next,mcost,mark,
    cost,found);}weight minimumSpanningArborescence(const graph&g,int r){const
    int n=sz(g);graph h(n);forn(u,n)forall(e,g[u])h[e->dst].pb(*e);vector<int>
    no(n);vector<vector<int>>comp(n);forn(u,n)comp[u].pb(no[u]=u);for(weight
    cost=0;;){vector<int>prev(n,-1);vector<weight>mcost(n,INF);forn(j,n)if(j!=
    r)forall(e,h[j])if(no[e->src]!=no[j])if(e->w<mcost[ no[j] ])mcost[ no[j]
    ]=e->w,prev[ no[j] ]=no[e->src];vector<vector<int>>next(n);forn(u,n)if(
    prev[u]>=0)next[ prev[u] ].push_back(u);bool stop=true;vector<int>mark(n);
    forn(u,n)if(u!=r&&!mark[u]&&!comp[u].empty()){bool found=false;visit(h,u,u

```

```
,r,no,comp,prev,next,mcost,mark,cost,found);if(found)stop=false;}if(stop){
forn(u,n)if(prev[u]>=0)cost+=mcost[u];return cost;}}}
```

## 7.15 Hungarian

```
1 //Dado un grafo bipartito completo con costos no negativos, encuentra el
  matching perfecto de minimo costo.
2 const tipo EPS=1e-9;const tipo INF=1e14;
3 #define N 502
4 tipo cost[N][N],lx[N],ly[N],slack[N];int n,max_match,xy[N],yx[N],slackx[N],
  prev2[N];bool S[N],T[N];void add_to_tree(int x,int prevx){S[x]=true,prev2[
  x]=prevx;forn(y,n)if(lx[x]+ly[y]-cost[x][y]<slack[y]-EPS)slack[y]=lx[x]+ly
  [y]-cost[x][y],slackx[y]=x;}void update_labels(){tipo delta=INF;forn(y,n)
  if(!T[y])delta=min(delta,slack[y]);forn(x,n)if(S[x])lx[x]-=delta;forn(y,n)
  if(T[y])ly[y]+=delta;else slack[y]-=delta;}void init_labels(){zero(lx),
  zero(ly);forn(x,n)forn(y,n)lx[x]=max(lx[x],cost[x][y]);}void augment(){if(
  max_match==n)return;int x,y,root,q[N],wr=0,rd=0;memset(S,false,sizeof(S)),
  memset(T,false,sizeof(T));memset(prev2,-1,sizeof(prev2));forn(x,n)if(xy[x
  ]==-1){q[wr++]=root=x,prev2[x]=-2;S[x]=true;break;}forn(y,n)slack[y]=lx[
  root]+ly[y]-cost[root][y],slackx[y]=root;while(true){while(rd<wr){x=q[rd
  ++];for(y=0;y<n;y++)if(cost[x][y]==lx[x]+ly[y]&&!T[y]){if(yx[y]==-1)break;
  T[y]=true;q[wr++]=yx[y],add_to_tree(yx[y],x);}if(y<n)break;}if(y<n)break;
  update_labels(),wr=rd=0;for(y=0;y<n;y++)if(!T[y]&&slack[y]==0){if(yx[y
  ]==-1){x=slackx[y];break;}else{T[y]=true;if(!S[yx[y]])q[wr++]=yx[y],
  add_to_tree(yx[y],slackx[y]);}if(y<n)break;}if(y<n){max_match++;for(int
  cx=x,cy=y,ty;cx!=-2;cx=prev2[cx],cy=ty)ty=xy[cx],yx[cy]=cx,xy[cx]=cy;
  augment();}}tipo hungarian(){tipo ret=0;max_match=0,memset(xy,-1,sizeof(xy
  ));memset(yx,-1,sizeof(yx)),init_labels(),augment();forn(x,n)ret+=cost[x][
  xy[x]];return ret;}
```

## 7.16 Dynamic Connectivity

```
1 struct UnionFind {
2     int n, comp;
3     vector<int> pre,si,c;
4     UnionFind(int n=0):n(n), comp(n), pre(n), si(n, 1) {
5         forn(i,n) pre[i] = i; }
6     int find(int u){return u==pre[u]?u:find(pre[u]);}
7     bool merge(int u, int v) {
8         if((u=find(u))==(v=find(v))) return false;
9         if(si[u]<si[v]) swap(u, v);
10        si[u]+=si[v], pre[v]=u, comp--, c.pb(v);
11        return true;
12    }
13    int snap(){return sz(c);}
14    void rollback(int snap){
15        while(sz(c)>snap){
16            int v = c.back(); c.pop_back();
```

```
17        si[pre[v]] -= si[v], pre[v] = v, comp++;
18    }
19 }
20 };
21 enum {ADD,DEL,QUERY};
22 struct Query {int type,u,v;};
23 struct DynCon {
24     vector<Query> q;
25     UnionFind dsu;
26     vector<int> match,res;
27     map<ii,int> last;//se puede no usar cuando hay identificador para cada
      arista (mejora poco)
28     DynCon(int n=0):dsu(n){}
29     void add(int u, int v) {
30         if(u>v) swap(u,v);
31         q.pb((Query){ADD, u, v}), match.pb(-1);
32         last[ii(u,v)] = sz(q)-1;
33     }
34     void remove(int u, int v) {
35         if(u>v) swap(u,v);
36         q.pb((Query){DEL, u, v});
37         int prev = last[ii(u,v)];
38         match[prev] = sz(q)-1;
39         match.pb(prev);
40     }
41     void query() {//podria pasarle un puntero donde guardar la respuesta
42         q.pb((Query){QUERY, -1, -1}), match.pb(-1);}
43     void process() {
44         forn(i,sz(q)) if (q[i].type == ADD && match[i] == -1) match[i] = sz(q);
45         go(0,sz(q));
46     }
47     void go(int l, int r) {
48         if(l+1==r){
49             if (q[l].type == QUERY)//Aqui responder la query usando el dsu!
50                 res.pb(dsu.comp);//aqui query=cantidad de componentes conexas
51             return;
52         }
53         int s=dsu.snap(), m = (l+r) / 2;
54         forr(i,m,r) if(match[i]!=-1 && match[i]<l) dsu.merge(q[i].u, q[i].v);
55         go(l,m);
56         dsu.rollback(s);
57         s = dsu.snap();
58         forr(i,l,m) if(match[i]!=-1 && match[i]>=r) dsu.merge(q[i].u, q[i].v);
59         go(m,r);
60         dsu.rollback(s);
61     }
```

## 8 Network Flow

### 8.1 Dinic

```

62 }dc;

1 // Corte minimo: vertices con dist[v]>=0 (del lado de src) VS. dist[v]==-1 (
  // del lado del dst)
2 // Para el caso de la red de Bipartite Matching (Sean V1 y V2 los conjuntos mas
  // proximos a src y dst respectivamente):
3 // Reconstruir matching: para todo v1 en V1 ver las aristas a vertices de V2
  // con it->f>0, es arista del Matching
4 // Min Vertex Cover: vertices de V1 con dist[v]==-1 + vertices de V2 con dist[v]
  // >0
5 // Max Independent Set: tomar los vertices NO tomados por el Min Vertex Cover
6 // Max Clique: construir la red de G complemento (debe ser bipartito!) y
  // encontrar un Max Independet Set
7 // Min Edge Cover: tomar las aristas del matching + para todo vertices no
  // cubierto hasta el momento, tomar cualquier arista de el
8 int nodes, src, dst;
9 int dist[MAX], q[MAX], work[MAX];
10 struct Edge {
11     int to, rev;
12     ll f, cap;
13     Edge(int to, int rev, ll f, ll cap) : to(to), rev(rev), f(f), cap(cap) {}
14 };
15 vector<Edge> G[MAX];
16 void addEdge(int s, int t, ll cap){
17     G[s].pb(Edge(t, sz(G[t]), 0, cap)), G[t].pb(Edge(s, sz(G[s])-1, 0, 0));}
18 bool dinic_bfs(){
19     fill(dist, dist+nodes, -1), dist[src]=0;
20     int qt=0; q[qt++]=src;
21     for(int qh=0; qh<qt; qh++){
22         int u=q[qh];
23         forall(e, G[u]){
24             int v=e->to;
25             if(dist[v]<0 && e->f < e->cap)
26                 dist[v]=dist[u]+1, q[qt++]=v;
27         }
28     }
29     return dist[dst]>=0;
30 }
31 ll dinic_dfs(int u, ll f){
32     if(u==dst) return f;
33     for(int &i=work[u]; i<sz(G[u]); i++){
34         Edge &e = G[u][i];

```

```

35         if(e.cap<=e.f) continue;
36         int v=e.to;
37         if(dist[v]==dist[u]+1){
38             ll df=dinic_dfs(v, min(f, e.cap-e.f));
39             if(df>0){
40                 e.f+=df, G[v][e.rev].f-= df;
41                 return df; }
42         }
43     }
44     return 0;
45 }
46 ll maxFlow(int _src, int _dst){
47     src=_src, dst=_dst;
48     ll result=0;
49     while(dinic_bfs()){
50         fill(work, work+nodes, 0);
51         while(ll delta=dinic_dfs(src,INF))
52             result+=delta;
53     }
54     // todos los nodos con dist[v]!=-1 vs los que tienen dist[v]==-1 forman el
55     // min-cut
56     return result; }

```

### 8.2 Min-cost Max-flow

```

1 const int MAXN=10000;
2 typedef ll tf;
3 typedef ll tc;
4 const tf INFFLUJO = 1e14;
5 const tc INFCOSTO = 1e14;
6 struct edge {
7     int u, v;
8     tf cap, flow;
9     tc cost;
10     tf rem() { return cap - flow; }
11 };
12 int nodes; //numero de nodos
13 vector<int> G[MAXN]; // limpiar!
14 vector<edge> e; // limpiar!
15 void addEdge(int u, int v, tf cap, tc cost) {
16     G[u].pb(sz(e)); e.pb((edge){u,v,cap,0,cost});
17     G[v].pb(sz(e)); e.pb((edge){v,u,0,0,-cost});
18 }
19 tc dist[MAXN], mnCost;
20 int pre[MAXN];
21 tf cap[MAXN], mxFlow;
22 bool in_queue[MAXN];

```

```

23 void flow(int s, int t) {
24     zero(in_queue);
25     mxFlow=mnCost=0;
26     while(1){
27         fill(dist, dist+nodes, INFCOSTO); dist[s] = 0;
28         memset(pre, -1, sizeof(pre)); pre[s]=0;
29         zero(cap); cap[s] = INFLUJO;
30         queue<int> q; q.push(s); in_queue[s]=1;
31         while(sz(q)){
32             int u=q.front(); q.pop(); in_queue[u]=0;
33             for(auto it:G[u]) {
34                 edge &E = e[it];
35                 if(E.rem() && dist[E.v] > dist[u] + E.cost + 1e-9){ // ojo EPS
36                     dist[E.v]=dist[u]+E.cost;
37                     pre[E.v] = it;
38                     cap[E.v] = min(cap[u], E.rem());
39                     if(!in_queue[E.v]) q.push(E.v), in_queue[E.v]=1;
40                 }
41             }
42         }
43         if (pre[t] == -1) break;
44         mxFlow +=cap[t];
45         mnCost +=cap[t]*dist[t];
46         for (int v = t; v != s; v = e[pre[v]].u) {
47             e[pre[v]].flow += cap[t];
48             e[pre[v]^1].flow -= cap[t];
49         }
50     }
51 }

```

## 9 Template

```

1 //touch {a..m}.in; tee {a..m}.cpp < template.cpp
2 #include <bits/stdc++.h>
3 using namespace std;
4 #define forr(i,a,b) for(int i=(a); i<(b); i++)
5 #define forn(i,n) forr(i,0,n)
6 #define sz(c) ((int)c.size())
7 #define zero(v) memset(v, 0, sizeof(v))
8 #define forall(it,v) for(auto it=v.begin();it!=v.end();++it)
9 #define pb push_back
10 #define fst first
11 #define snd second
12 typedef long long ll;
13 typedef pair<int,int> ii;
14 #define dforr(i,n) for(int i=n-1; i>=0; i--)

```

```

15 #define dprint(v) cout << #v"=" << v << endl //;)
16
17 const int MAXN=100100;
18 int n;
19
20 int main() {
21     freopen("input.in", "r", stdin);
22     ios::sync_with_stdio(0);
23     while(cin >> n){
24
25     }
26     return 0;
27 }

```

## 10 Ayudamemoria

Leer hasta fin de linea

```

1 #include <sstream>
2 //hacer cin.ignore() antes de getline()
3 while(getline(cin, line)){
4     istringstream is(line);
5     while(is >> X)
6         cout << X << "□";
7     cout << endl;
8 }

```

Expandir pila

```

1 #include <sys/resource.h>
2 rlimit rl;
3 getrlimit(RLIMIT_STACK, &rl);
4 rl.rlim_cur=1024L*1024L*256L;//256mb
5 setrlimit(RLIMIT_STACK, &rl);

```

Iterar subconjunto

```

1 for(int sbm=bm; sbm; sbm=(sbm-1)&bm)

```